NECパーソナルコンピュータ
PC-9800シリーズ

NEC

Software library

# MS-DOS™ 5.0

## プログラム開発ツールマニュアル



Software library

# MS-DOS™ 5.0

## プログラム開発ツールマニュアル

ご注意

(1) 本書の内容の一部または全部を、無断で他に転載することは禁止されています。
(2) 本書の内容は、将来予告なしに変更することがあります。
(3) 本書の内容は、万全を期して作成しております。万一、ご不審な点や誤り、記載もれなどお気づきの点がありましたら、ご連絡ください。
(4) 運用した結果の影響については、(3)項にかかわらず責任を負いかねますのでご了承ください。

Microsoft（マイクロソフト）とそのロゴは米国マイクロソフト社の登録商標です。
MASM は米国マイクロソフト社の商標です。
MS-DOS は米国マイクロソフト社の登録商標です。
XENIX は米国マイクロソフト社の商標です。
Intel（インテル）は米国インテル社の商標です。
80286、386、386SX、486 は米国インテル社の商標です。



輸出する際の注意事項

本製品（ソフトウェア）は日本国内仕様であり、外国の規格等には準拠しておりません。
本製品を日本国外で使用された場合、当社は一切責任を負いかねます。また、当社は本製品に関して、海外での保守サービスおよび技術サポート等は行っておりません。



AB1A1A

# はじめに

本書は、「MS-DOS 拡張機能セット」中の「MS-DOS プログラム開発ツールディスク」に収められているユーティリティプログラムについて解説したものです。

## 本書の目的と構成

本書の目的は、プログラム開発のためのユーティリティプログラムの使用法について理解していただくことにあります。したがって、本書をお読みになる場合、プログラム開発に関する基礎的な知識を習得されていることが前提となります。

### ■第１章「プログラム開発ツールの利用にあたって」

この章は、お求めになった「MS-DOS 拡張機能セット」中の「MS-DOS プログラム開発ツールディスク」のファイル構成、MASM を用いたプログラム開発の手順、各ユーティリティの用途、本書中で用いる表記法などについて説明しています。

### ■第２章「LINK：リンカ」

LINK は、オブジェクトファイル内のコードとデータを結合するとともに、指定されたライブラリを探索して外部参照を解決し、リロケータブル（再配置可能）な実行イメージとリロケート（再配置）情報で、実行可能なファイルを作成するものです。この章では、LINK の起動法、使用可能なスイッチ、マップファイルの利用法などについて説明しています。

### ■第３章「MAPSYM：シンボルマップユーティリティ」

MAPSYM は、LINK が作成したマップファイルを利用して、SYMDEB 用のシンボルファイルを作成するものです。この章では、その作成方法について説明しています。

## ■第４章「SYMDEB：シンボリックデバッガ」

SYMDEB は、メモリやレジスタの操作、プログラムのトレースや逆アセンブルといった一般的なデバッガの機能に加えて、サブルーチンや変数などをシンボル(名前)で参照したり、ソースの表示や番号付けをしたりすることのできるシンボリックデバッガです。この章では、SYMDEB の起動法と、デバッグ中に使用できるすべてのコマンドについて説明しています。

## ■第５章「LIB：ライブラリマネージャ」

LIB は、ライブラリファイル中に、オブジェクトファイルを追加／登録したり、入れ換えたり、削除したりするために用いるユーティリティです。この章では、LIB の起動法と、ライブラリ管理のために使用できるコマンドについて説明しています。

## ■第６章「MAKE：プログラムメインテナ」

MAKE は、マクロアセンブラや高級言語によるプログラムの開発工程の保守／管理を、単なるバッチ処理より高度に行うためのユーティリティです。この章では、メイクファイルの作成法、MAKE の起動法、動作例などについて説明しています。

## ■第７章「EXE2BIN：バイナリファイルコンバータ」

EXE2BIN は、EXE 形式のプログラムファイルに加工を施して、COM 形式のプログラムファイルを作成したり、絶対アドレス上に置かれるプログラムファイルを作成したりするのに使うユーティリティです。この章では、EXE2BIN の起動法と使い方について説明しています。

## ■第８章「DEBUG：デバッガ」

DEBUG は、メモリやレジスタの操作、プログラムのトレースや逆アセンブルを行うだけでなく、EMS メモリ(拡張メモリ)のステータス表示やマッピングを行ったりすることのできるデバッガです。DEBUG では、EMS 関連のコマンドを除いたすべてのコマンドは、SYMDEB の同名のコマンドと同じ機能を持っていますので、それらのコマンドについては第 4 章「SYMDEB：シンボリックデバッガ」をご覧ください。この章では、DEBUG の起動法と、DEBUG のみに備えられている EMS メモリ関連のコマンドについて説明しています。

# その他のマニュアル

「MS-DOS 拡張機能セット」には、本書の他に次のようなマニュアルが添付されています。

## ■『MS-DOS ユーザーズリファレンスマニュアル』

システムディスクに収められているMS-DOSのすべてのコマンドについて、詳しく説明しています。また、「MS-DOS 基本機能セット」では扱われていない、MS-DOSの高度な機能についても解説しています。MS-DOSの手引きとして、ご利用ください。

## ■『日本語入力ガイド』

MS-DOS上で利用可能な日本語入力機能について解説しています。日本語の入力を行う方法と、その他の有用な機能について詳しく説明し、また、辞書ファイルを保守管理するユーティリティ(DICM)や、ユーザーが独自の記号や漢字を作成して利用するためのユーティリティ(USKCGM)についても説明しています。

## ■『プログラマーズリファレンスマニュアル Vol.1』
## ■『プログラマーズリファレンスマニュアル Vol.2』

MS-DOSの内部的な技術情報を、詳細に説明しています。Vol.1では、MS-DOSの提供する各種機能(システムコール、ファンクションコール)や、プログラムおよびメモリ管理に関する技術情報を扱っています。Vol.2では、周辺装置を制御するデバイスドライバについての情報を扱っています。MS-DOSの内部機能を使用するプログラムを作成される際にご利用ください。

# 目　次

## 第5章 LIB：ライブラリマネージャ 105



## 第6章 MAKE：プログラムメインテナ 121



## 第7章 EXE2BIN：バイナリファイルコンバータ 127

## 第8章 DEBUG：デバッガ 131

# 第 1 章

# プログラム開発ツールの利用にあたって

本書「プログラム開発ツールマニュアル」では、プログラム開発で利用する各種ユーティリティの解説を行います。開発言語としては、マイクロソフトマクロアセンブラ：MASMを想定して記述しています。

この第1章では、各種ユーティリティの概要と本書で用いている表記法などについて解説します。

## 1.1 ファイル構成

本書で、解説するユーティリティはつぎのとおりです。

| ファイル名 | 機　能 |
|---|---|
| LINK.EXE | リンカ |
| MAPSYM.EXE | シンボルマップユーティリティ |
| SYMDEB.EXE | シンボリックデバッガ |
| LIB.EXE | ライブラリマネージャ |
| MAKE.EXE | プログラムメインテナ |
| EXE2BIN.EXE | バイナリファイルコンバータ |
| DEBUG.EXE | デバッガ |

## 1.2　MASM の利用にあたって

MASM は、8086 マイクロプロセッサファミリー用のマクロアセンブリ言語です。対象となるプロセッサは 8086、80286、386、486 などをはじめとして、8087、80287 の数値演算コプロセッサ用のプログラムも開発することができます。

MASM でプログラムを開発するには、マクロアセンブリ言語の文法とソースファイルの書式などの予備知識を必要とします。また、8086/80286/386/486 マイクロプロセッサファミリーの機能と命令(インストラクション)セットに関する知識も必要です。

MASM には、一般的な疑似命令に加えて、型を持ったデータを扱ったり、セグメントを処理するための疑似命令が用意されています。これによって、8086 マイクロプロセッサのアーキテクチャを論理的に扱うことができます。MASM の文法、疑似命令(ディレクティブ)、オペランド、用例などについては、「マクロアセンブラ　ユーザーズ／リファレンスマニュアル」を参照してください。

**注意**　　MASM と「マクロアセンブラ　ユーザーズ／リファレンスマニュアル」は、別売のマクロアセンブラパッケージに添付されています。

## 1.3　ユーティリティの用途

アセンブリプログラムのソースファイルの作成から、実行可能なプログラムが完成するまでの手順は、つぎのような 4 段階に分けられます。

**1. ソースファイルの作成**

エディタを使ってアセンブリプログラムのソースファイルを作ります。



**2. ソースファイルのアセンブル**

MASM(マクロアセンブラ)でソースプログラムをアセンブルし、オブジェクトコードを生成します。



**3. オブジェクトコードのリンク**

LINK(リンカ)で、別に開発したプログラムのオブジェクトコードやライブラリファイルのルーチンと結合し、実行可能なプログラムを生成します。



**4. プログラムの検証・テスト**

プログラムを実行し、望みの結果が得られなければ、SYMDEB(シンボリックデバッガ)、または DEBUG(デバッガ)でプログラムを検証／テストします。

ユーティリティは、以上のようにしてプログラムを作成する場合の各種作業を支援するもので、名称および用途はつぎのとおりです。

・MAKE（プログラムメインテナ）

プログラムを開発するときは、一般に何度かアセンブル、リンク、テストを繰り返します。MAKEは、このような煩雑な作業を自動化するためのユーティリティです。

・MAPSYM（シンボルマップユーティリティ）

LINKの生成したマップファイルから、SYMDEBでデバッグするためのシンボルマップファイルを作成します。

・LIB（ライブラリマネージャ）

リンクの際に用いるライブラリファイルを管理します。ユーザー独自のライブラリファイルを作成したり、ライブラリファイルの内容を変更するときに用います。

・EXE2BIN（バイナリファイルコンバータ）

完成したEXE形式のプログラムファイルを、条件が揃えばCOM形式のファイルに変換します。また、プログラムを絶対アドレスにロードするための変換も行うことができます。

## 1.4 本書で用いる表記法

このマニュアルでは、つぎのような表記法を用いています。

| 表記 | 意　味 |
|---|---|
| CAPS | **大文字のアルファベット**は、表記されているとおりに入力する、コマンド名、パラメータなどを表します。なお、通常は大文字／小文字はどちらでも同様に扱われます。 |
| 〈 〉 | **山形カッコ**は、その位置に入力する項目を表します。たとえば、〈ファイル名〉には、ファイルの名前を入力します。 |
| [ ] | **角形カッコ**は、入力を省略できる項目を表します。省略した場合は、システムであらかじめ設定された（デフォルトの）処理が行われます。 |
| \| | **縦線**は、選択項目の区切りを表します。この記号で区切られた項目の中から、必要なものを選んで入力します。 |
| … | **繰り返し記号**は、必要に応じて、項目を繰り返し入力することを表します。 |
| KEY | 枠で囲まれた文字は、キーボード上の特定のキーを示します。たとえば、CTRL はコントロールキーを表します。また、CTRL + C のような表記は、CTRL キーを押しながら C キーを押すことを表します。 |

カンマ（,）、コロン（：）、セミコロン（；）、スラッシュ（／）、イコール（=）などの記号類とスペース（空白）は、コマンドなどの書式の一部です。表記されている位置に正確に入力してください。

# 第 2 章

# LINK：リンカ



## 2.1 イントロダクション

リンカ：LINK は、MASM または C、Pascal のようなコンパイラによって生成されたオブジェクトコードから、MS-DOS で実行可能なファイル（EXE 形式）を作成します。

LINK を使用するときは、オブジェクトファイルのファイル名と必要なライブラリファイルを指定します。LINK はオブジェクトファイル内のコードとデータを結合し、指定されたライブラリを検索して外部参照を解決します。次に LINK は、リロケータブル（再配置可能）な実行イメージとリロケート（再配置）情報で、実行可能なファイルを作成します。

MS-DOS は、LINK が生成したリロケート情報を使用して、適切なメモリアドレスに実行イメージをロードし実行します。

LINK は、最大 1M バイトまでのプログラムを処理することができます。

なお本書では、従来バージョンと同じ機能についてのみ解説しています。

## 2.2 LINK の起動と使い方

LINK を使うときは、オブジェクトファイルのファイル名、ライブラリファイル名、スイッチなどのパラメータを指定します。パラメータの指定方法にはつぎの 3 種類あります。

1. コマンドラインでの指定方法
2. LINK のプロンプトでの指定方法
3. 応答ファイルでの指定方法

いずれの方法で LINK を起動した場合でも、[CTRL]＋[C] キーを押すことでいつでも処理を中止することができます。つぎに、それぞれの指定方法で LINK を起動する方法を説明します。

## ■ コマンドラインでの指定方法

このコマンドの書式は、つぎのとおりです。

LINK　<オブジェクトファイル名>[+<オブジェクトファイル名>…],
[<出力ファイル名>],[<マップファイル名>],
[<ライブラリファイル名>+<ライブラリファイル名>…]
[<スイッチ>…][；]

＜オブジェクトファイル名＞

リンクしたいオブジェクトファイルのファイル名を指定します。少なくとも1つのファイル名を指定しなければなりません。拡張子をつけずに指定した場合、〝.OBJ〞がついているものとして処理されます。複数のファイルを指定するときは、スペースまたはプラス記号(+)で区切って指定します。

＜出力ファイル名＞

実行可能ファイルの名前を指定します。省略すると、最初のオブジェクトファイル名の拡張子を〝.EXE〞に変えたファイル名で作成します。また、拡張子なしのファイル名を指定した場合、その名前に自動的に〝.EXE〞がつけられます。

＜マップファイル名＞

マップ情報を格納するファイルの名前を指定します。拡張子をつけずに指定した場合、〝.MAP〞が自動的につけられます。ファイルの名前を指定せず、/MAP、/LINENUMBERS スイッチも指定しなかった場合、マップファイルは作成されません。

＜ライブラリファイル名＞

オブジェクトファイルとリンクしたいルーチンの入っているライブラリの名前を指定します。拡張子をつけずに指定した場合、〝.LIB〞がついているものとして処理されます。ライブラリ名を指定しない場合は、セミコロン(；)を入力してください。なお、複数のライブラリを指定したり、ライブラリファイルの検索パスを設定することもできます。詳しくは2.2節中の「ライブラリファイルの検索パスの設定」を参照してください。

＜スイッチ＞

LINK の動作を制御するもので、必要に応じて指定します。スイッチはコマンドライン上のどこにでも入れることができます。スイッチの内容については2.3「LINK のスイッチ」を参照してください。

コマンドラインの任意の場所に〝；〃を入力すると、それ以降のファイル名の指定が無視され、デフォルトのファイル名が用いられます。

**例：** LINK test；

これはもっとも単純な例です。LINKは、〝test.obj〃というオブジェクトファイルをリンクして、〝test.exe〃という実行可能ファイルを作成します。マップファイルは作成せず、またライブラリの検索も行いません。

特定のファイル名が指定されない場合でも、つぎの例のようにファイル名を区切るカンマ(,)が必要な場合があります。

**例：** LINK test,,,io.lib

この例では、〝test.obj〃というオブジェクトファイルと〝io.lib〃というライブラリファイルから必要なルーチンをリンクして、〝test.exe〃という実行可能ファイルを作成します。マップファイルは作成しません。

**例：** LINK startup+file,file,file；

この例では〝startup.obj〃および〝file.obj〃という2つのオブジェクトファイルから、〝file.exe〃という名前の実行可能ファイルを作成します。さらに〝file.map〃というマップファイルも作成します。ライブラリは検索しません。

**例：** LINK moda+modb+modc+startup/PAUSE,,abc,¥lib¥math

この例では〝moda.obj〃、〝modb.obj〃、〝modc.obj〃および〝startup.obj〃という4つのオブジェクトモジュールがリンクされ、さらに〝¥lib〃ディレクトリ内の〝math.lib〃というライブラリファイルが検索されて必要なルーチンがリンクされます。そして、〝moda.exe〃という名前の実行可能ファイルと、〝abc.map〃という名前のマップファイルが作成されます。なお、/PAUSEスイッチが指定されているので、実行可能ファイルを作成する前に処理を一時中断します。

## ■ LINKのプロンプトでの指定方法

コマンドラインで〝LINK〃のみを入力すると、LINKは必要な情報の入力を求めるプロンプトを表示します。このプロンプトに応えて、各種のファイルやスイッチを指定します。指定方法は、つぎのとおりです。

1. LINKとタイプし、リターンキーを押すと、つぎのようなプロンプトが表示されます。

   Object Modules [.OBJ]：

このプロンプトには、オブジェクトファイルのファイル名を指定します。省略はできません。複数のファイルを指定するときは、スペースまたはプラス記号(+)で区切ります。拡張子についてはコマンドラインで指定する方法とまったく同じです。また、指定するオブジェクトファイルが1行で入りきらない場合は、その行の最後にプラス記号(+)をタイプしてリターンキーを押します。こうすると、さらに他のオブジェクトファイルの指定を促すプロンプトが表示されます。

2. オブジェクトファイルを指定したら、リターンキーを押します。画面には、つぎのプロンプトが表示されます。

   Run File [filename.EXE]：

   このプロンプトには、実行可能ファイルのファイル名を指定します。省略した場合や拡張子の扱いはコマンドラインで指定する方法とまったく同じです。

3. 実行可能ファイルを指定したら、リターンキーを押します。画面には、つぎのプロンプトが表示されます。

   List File[NUL.MAP]：

   このプロンプトには、マップファイルのファイル名を指定します。省略した場合や拡張子の扱いはコマンドラインで指定する方法とまったく同じです。

4. 最後にライブラリファイルのファイル名を入力するように、つぎのプロンプトが表示されます。

   Libraries[.LIB]：

   これもコマンドラインで指定する方法と同じです。なお、1行で入りきらない場合は、その行の最後にプラス記号(+)をタイプしてリターンキーを押します。こうすると、さらに他のライブラリファイルの指定を促すプロンプトが表示されます。

スイッチは、どの行に入れてもかまいません。また、任意のプロンプトの後に〝；〞を入力すると、それ以降のすべての指定を省略したものとして処理します。

**例：**

```
Object  Modules[.OBJ]：moda+modb+modc+modee+
Object  Modules[.OBJ]：startup/PAUSE
Run  File[moda.EXE]：
List  File[NUL.MAP]：abc
Libraries[.LIB]：¥lib¥math
```

この例は、コマンドラインで

LINK moda＋modb＋modc＋modee＋startup/PAUSE,,abc,¥lib¥math

と入力したものと同じです。

## ■ 応答ファイルでの指定方法

応答ファイル内に、処理に関係したすべてのファイルの名前を記述し、コマンドラインでその応答ファイルを指定します。この場合は、つぎのように〝@″をつけて指定します。

LINK　@<応答ファイル名>

応答ファイルには、好きなファイルの名前をつけることができます。このファイルの内容は、つぎのような形で記述します。

<オブジェクトファイル名>
[<出力ファイル名>]
[<マップファイル名>]
[<ライブラリファイル名>]

応答ファイル内の各行は、前項で解説したプロンプトの入力に対応しています。指定する名前が1行に入り切らない場合は、その行の最後にプラス記号(＋)を入力して、次の行に継続することができます。ファイル名を指定しない場合は、その行を空にしておきます。

スイッチは、どの行でも指定可能です。応答ファイル内のどの行でも、〝；″を入れると、それ以降の指定は無視されます。

応答ファイルに、必要なファイル名が入っていない場合、LINKは不足しているファイル名の入力を求めるコマンドプロンプトを表示し、応答がキーボードから入力されるまで待ちます。

**例：**

```
moda modb modc startup
/PAUSE
abc
¥lib¥math
```

この応答ファイルは、コマンドラインでつぎのように入力した場合と同じです。

LINK moda modb modc startup/PAUSE,,abc,¥lib¥math

## ■ ライブラリファイルの検索パスの設定

LINKは、ライブラリファイルを検索するためのパスを環境変数〝LIB″に設定しておくと、それにしたがってライブラリを検索します。複数のパスを検索する場合は、〝；″で区切ります。最大16個までのパスを指定できます。検索パスは、ライブラリに明示のパス指定がない場合のみ使用されます。たとえば、

```
SET LIB = A：¥lib；B：¥system¥lib
```

というコマンドで検索パスを設定した後、

```
LINK file,,file.map,math＋common
```

を実行すると、〝math.lib″および〝common.lib″というライブラリを見つけるために、まずカレントディレクトリ、続いてドライブA上の〝¥lib″、ドライブB上の〝¥system¥lib″の順にライブラリを検索します。

検索パスは、コマンドラインで指定することもできます。この場合も、検索パスは明示のパス指定がない場合のみ使用されます。

**例：**　LINK　file,,file.map,A：¥altlib¥math.lib＋common＋B：＋D：¥lib¥

この例では、〝math.lib″というライブラリを見つけるために、ドライブA上の〝¥altlib″のみを検索しますが、〝common.lib″を見つけるためには、カレントドライブのカレントディレクトリ、ドライブB上のカレントディレクトリ、および最後にドライブD上の〝¥lib″というディレクトリを検索します。

## ■ 一時ディスクファイル――VM.TMP

LINKは使用可能なメモリを使いつくした場合、カレントディレクトリ内に〝VM.TMP″という一時ファイルを作成します。このファイルを作成したときには、つぎのようなメッセージを表示します。

```
VM.TMP has been created.
Do not change diskette in drive,<d：>
```

このメッセージが表示された後は、処理が終了するまでこのディスクを抜かないでください。処理が終わると、この一時ファイルは自動的に削除されます。

**注意**　ユーザー自身のファイルとして〝VM.TMP″というファイル名を使用しないようにしてください。LINKが一時ファイルを作成する際に、ユーザー自身のファイルを書き換えてしまいます。

# 2.3　LINK のスイッチ

LINK のスイッチは、コマンドラインのどこにでも指定できます。スイッチには、つぎのようなものがあります。

| スイッチ | 省略形 | 機　能 |
|---|---|---|
| /PAUSE | /P | リンク中の中断 |
| /MAP | /M | パブリックシンボルマップの作成 |
| /STACK:〈n〉 | /ST:〈n〉 | スタックサイズの設定 |
| /CPARMAXALLOC:〈n〉 | /C:〈n〉 | 最大割り当てスペースの設定 |
| /HIGH | /H | 上位開始アドレスの設定 |
| /DSALLOCATE | /DS | データグループの割り当て |
| /LINENUMBERS | /LI | 行番号の付加 |
| /NOIGNORECASE | /NOI | 大文字小文字の区別 |
| /NODEFAULTLIBRARYSEARCH | /NOD | 省略時のライブラリの無視 |
| /NOGROUPASSOCIATION | /NOG | プログラム内のグループの無視 |
| /OVERLAYINTERRUPT:〈n〉 | /O:〈n〉 | オーバレイ割り込みの設定 |
| /SEGMENTS:〈n〉 | /SE:〈n〉 | 最大のセグメントの設定 |
| /DOSSEG | /DO | 標準のセグメント配置の利用 |

以降、各スイッチについて解説しますが、スイッチの表記には省略形を使います。

## ■ リンク中の中断

**書　式**　/P

**解　説**

/P スイッチを指定すると、実行可能ファイルを作成する前に、つぎのメッセージを表示して中断します。

```
About to generate.EXE file
Change diskette in drive＜d：＞and press＜ENTER＞
```

これによって、実行可能ファイルを書き込む前にディスクを交換することができます。リターンキーを押すと、処理を再開します。

**注意**　VM.TMP ファイルが作成されている場合、このファイルのために使用されているディスクを取り外してはいけません。

## ■ パブリックシンボルマップの作成

**書　式**

/M

**解　説**

マップファイルのファイル名だけを指定した場合、セグメントの名前／長さ／ロードアドレス、グループの名前／ロードアドレス／エントリエドレスが書き込まれます。/M スイッチを指定した場合、これらの情報に加えて、すべてのパブリックシンボルの情報も書き込みます。

マップファイルについては、2.4「マップファイル」をご覧ください。

**注意**　マップファイルを指定しなかった場合でも、/M スイッチを指定するとマップファイルが自動的に作成されます。このファイル名は、先頭のオブジェクトファイルと同一のファイル名に〝.MAP〟という拡張子がつけられたものとなります。

## ■ スタックサイズの設定

**書　式**

/ST：<n>

**解　説**

LINK は、通常、オブジェクトファイルにあるスタックセグメントの大きさに基づいて、スタックサイズを自動的に計算します。/ST スイッチを指定した場合、計算する代わりにスタックを<n>によって指定されたバイト数にセットします。<n>は、1〜65535 の範囲内で 10 進、8 進、または 16 進数で指定できます。なお 8 進数は〝0〟、16 進数は〝0x〟で始めなければなりません。

**例：**　/ST：0xFF

この例では、スタックサイズを 255(FFH)バイトに設定します。

## ■ 最大割り当てスペースの設定

**書　式**　/C：<n>

**解　説**

LINKは、通常、プログラムに必要なエリアを最大のパラグラフ数(1パラグラフ=16バイト)である65535にセットします。これはすべてのメモリを表していることになるため、MS-DOSはプログラムをロードしたときに常にこの要求を拒否し、存在する最大の連続したブロックのメモリを割り当てることになります。

/Cスイッチを指定した場合、MS-DOSはこの<n>で指定された量のスペースを割り当てます。これは、メモリ中の他のスペースを他のプログラムが使用できることを意味します。<n>が必要とされる最小のパラグラフ数より小さい場合、LINKはこの指定を無視し、必要とされる最小のパラグラフ数と等しくなるように割り当てスペースをセットします。プログラムによって必要とされる最小のパラグラフ数は、常にこのプログラム中のコードおよびデータのパラグラフ数以上になります。

<n>にはパラグラフ数を指定します。<n>は、1〜65535の範囲内で10進、8進、または16進数で指定できます。なお8進数は〝0〟、16進数は〝0x〟で始めなければなりません。

## ■ 上位開始アドレスの設定

**書　式**　/H

**解　説**

/Hスイッチを指定した場合、プログラムの開始アドレスは使用可能なメモリのできるだけ高いアドレスにセットされます。このスイッチがない場合、プログラムの開始アドレスはメモリ中のできるだけ低いアドレスにセットされます。

## ■ データグループの割り当て

**書　式**　/DS

**解　説**

通常、LINKは〝DGROUP〟というグループ内の最下位バイトにオフセット0000Hを割り当てますが、/DSを指定した場合は、このグループ内の最上位バイトにオフセットFFFFHを割り当てます。この結果、出力されたデータは、DGROUPが入っているメモリセグメント中のできるだけ高いアドレスにロードされます。

/DSスイッチは、プログラムの開始点より低いアドレスの未使用のメモリを利用するために、通常/Hスイッチといっしょに使用されます。LINKは、DGROUP内のすべての使用可能なエリアは、このプログラムより低い隣接したアドレスのメモリを占有していると想定します。このグループを使用するために、セグメントレジスタはDGROUPの開始アドレスにセットされなければなりません。

## ■ 行番号の付加

**書　式**　/LI

**解　説**

/LI スイッチを指定した場合、マップファイルに行番号を付加します。ただし、これはオブジェクトファイルに行番号情報が入っていなければなりません(MASM や一部の高級言語コンパイラでは、行番号情報を入れることはできません)。オブジェクトファイルに行番号情報がない場合、/LI スイッチは無視されます。

**注意**　マップファイルを指定しなかった場合でも、/LI スイッチを指定するとマップファイルが自動的に作成されます。このファイル名は先頭のオブジェクトファイルと同一のファイル名、および".MAP"という拡張子をつけます。

## ■ 大文字小文字の区別

**書　式**　/NOI

**解　説**

/NOI スイッチを指定した場合、LINK はシンボル名の大文字と小文字を区別して扱います。通常、LINK は大文字と小文字を同一のものと見なし処理します。

/NOI スイッチは、大文字と小文字を区別する高級言語のコンパイラが作成したオブジェクトファイルに対して使用します。

## ■ 省略時のライブラリの無視

**書　式**　/NOD

**解　説**

高級言語のコンパイラは、ライブラリと確実にリンクされるように、ライブラリ名をオブジェクトファイルに付加することができます。/NOD スイッチを指定した場合、このライブラリ名を無視して、LINK のコマンドラインで指定したライブラリのみ検索するようになります。

## ■ プログラム内のグループの無視

**書　式**　/NOG

**解　説**

/NOG スイッチを指定した場合、データやコードの項目にアドレスを割り当てるとき、グループの関連を無視するようになります。

なお、このスイッチの使用は、あまりおすすめできません。

## ■ オーバーレイ割り込みの設定

**書　式**

/O：<n>

**解　説**

/O スイッチを指定した場合、オーバーレイローディングルーチンの割り込み番号が＜n＞にセットされます。このスイッチによって、デフォルトのオーバーレイ割り込み番号(03FH)が無効になります。

＜n＞は、0～255 の範囲内で、10 進、8 進、または 16 進数で指定できます。なお 8 進数は〝0〞、16 進数は〝0x〞で始めなければなりません。

**注意**　MS-DOS やデバイスドライバなどの割り込みと競合しない割り込み番号を使ってください。

## ■ 最大のセグメント数の設定

**書　式**

/SE：<n>

**解　説**

/SE スイッチは、LINK が処理するプログラムのセグメントの数を設定します。LINK は、デフォルトで 128 までのセグメントを処理しますが、/SE スイッチでこのセグメント数を設定できます。プログラムが 128 より多くのセグメントによって構成されている場合に使用します。指定された値より多くのセグメントを検出した場合、エラーメッセージを表示して停止します。

＜n＞は、1～1024 の範囲内で、10 進、8 進、または 16 進数で指定できます。なお 8 進数は〝0〞、16 進数は〝0x〞で始めなければなりません。

## ■ 標準のセグメント配置の利用

**書　式**

/DO

**解　説**

/DO を指定した場合、MS-DOS のセグメントの順序付けの規約に従って、実行可能ファイル内のセグメントを配置します。この規約は、つぎのとおりです。

1. 〝CODE〞というクラス名が付けられたすべてのセグメントは、実行可能ファイルの開始点に置かれる。
2. 〝DGROUP〞という名前のグループに属さないすべてのセグメントは、〝CODE〞セグメントの直後に置かれる。
3. 〝DGROUP〞に属するセグメントは、このファイルの終わりに置かれる。

## 2.4　マップファイル

マップファイルには、プログラム中のセグメントの情報とグループの情報、検出される可能性のあるエラーメッセージが書き込まれます。

セグメント情報はつぎのような形式です。

```
Strat     Stop      Length    Name    Class
00000H    0172CH    0172DH    TEXT    CODE
01730H    01E19H    006EAH    DATA    DATA
```

〝Start″および〝Stop″の項には、各セグメントの先頭および最終バイトの20ビットアドレス(16進)が示されています。このアドレスは、00000Hからの相対アドレスです。〝Length″の項にはセグメントの長さがバイト単位で、〝Name″の項にはセグメントの名前が、〝Class″の項にはセグメントのクラス名が示されます。

グループ情報は、つぎのような形式になっています。

```
Origin    Group
0000：0   IGROUP
0173：0   DGROUP
```

この例では、IGROUPはコード(命令)グループの名前、DGROUPはデータグループの名前です。マップファイルの終わりに、LINKはプログラムのエントリポイントのアドレスを示します。

/Mスイッチを指定した場合、上記のマップファイルにパブリックシンボルを付加します。これらのシンボルは、2回出力され、1回目はアルファベット順、次はロードアドレス順に示されます。

例：

```
ADDRESS           PUBLICS BY NAME
0000：1567   brk
0000：1696   chmod
0000：01DB   chkstk
0000：131c   clearerr
0173：0035   fac

ADDRESS           PUBLICS BY NAME
0000：01DB   chkstk
0000：131C   clearerr
0000：1567   brk
0000：1696   chmod
0173：0035   fac
```

パブリックシンボルのアドレスは、セグメント：オフセットという形式で示されます。これは、ロードモジュールの開始点(0000：0000)からの相対位置を示しています。

/Hおよび/DSスイッチを指定し、リンクしたものが64Kバイト以下であるとき、大きな値のセグメントアドレスを持つシンボルが示されます。このアドレスは、シンボルがコードおよびデータの実際の開始点より低い位置にあることを示しています。

**例：** FFF0：0A20 template

この例では、〝template〟はこのプログラムの開始点より低い位置にあることが示されています。実際のtemplateの20ビットアドレスが、00920Hであることに注意してください。

## 2.5 LINKの動作方法

SEGMENTやGROUPのMASMの疑似命令によって、LINKが作成する実行可能ファイルが変化します。この節では、LINKのセグメントの結合方法と参照を解決するための処理方法を説明します。

### ■ セグメントの位置合わせ

SEGMENT疑似命令のセグメントの位置合わせのタイプ(アラインメント)にはBYTE、WORD、PARA、PAGEがあります。これらは、バイト、ワード、パラグラフ、およびページの境界と対応しており、それぞれ、1、2、16、および256の倍数のアドレスを表しています。

LINKはセグメントを検出したとき、この位置合わせのタイプをチェックします。そこでBYTEタイプのときは直前のセグメントとすきまなく詰め、WORD、PARA、PAGEである場合、最後にコピーされたバイトが該当する境界で終了しているかチェックし、そうでない場合には境界までをゼロで埋めます。

### ■ フレーム番号

LINKは、プログラム中の各セグメントごとに、開始アドレスを計算します。この開始アドレスは、セグメントの位置合わせ、および実行可能ファイルにすでにコピーされたセグメントの大きさに基づいて行われます。このアドレスは、オフセットおよび〝標準フレーム番号〟によって構成されます。標準フレーム番号は、1バイト以上のバイトのセグメントが入っているメモリ中の先頭のパラグラフのアドレスを指します。フレーム番号は、常に16の倍数(パラグラフのアドレス)です。オフセットは、このパラグラフの開始点からセグメント内の先頭バイトまでのバイト数です。BYTEおよびWORDの位置合わせの場合、オフセットはゼロでない場合がありますが、PARAおよびPAGEの位置合わせの場合は常にゼロです。

セグメントをリンクするとき、このセグメントのフレーム番号をLINKによって作成されたマップファイルから取得することができます。フレーム番号は、このセグメントのために指定された〝開始〟アドレスの16進の先頭の5桁です。

## ■ セグメントの順序

LINK は、オブジェクトファイル内でセグメントを検出した順序で、実行可能ファイルを作成します。この順序は、LINK が同一のクラス名が付けられた複数のセグメントを検出しないかぎり、このプログラムを通して維持されます。同一のクラス名が付けられたセグメントは同一のクラスに属し、連結したブロックとして実行可能ファイルにコピーされます。

つぎのプログラム例では、〝DATAX〟および〝DATAZ〟というセグメントは1クラスを形成します。両方のセグメントは、実行可能ファイルで〝TEXT〟セグメントの前に作成されます。

例：

```
DATAX    segment    'DATA'
DATAX    ends

TEXT     segment    'CODE'
TEXT     ends

DATAZ    segment    'DATA'
DATAZ    ends
```

すべてのセグメントは、何らかのクラスに属します。クラス名が明示で定義されていないセグメントには、〝null〟クラス名が付けられ、null クラス名が付けられた他のセグメントと一緒に連続したブロックとして処理されます。

LINK では、1クラス内のセグメント数や大きさは無制限です。1クラス内のすべてのセグメントを総計した大きさが、64K バイト以上であっても構いません。

## ■ セグメントの結合

LINK は、結合タイプ(コンバイン)によって、同一のセグメント名を共有する複数のセグメントを単一の大規模なセグメントに結合すべきかどうかを決定します。結合タイプには、public、stack、memory、common、private があります。

結合タイプが public である場合、LINK は同一の名前で同一のクラスに属するすべてのセグメントを結合します。セグメントを結合するとき、これらのセグメントが連続しており、これらのセグメント内のすべてのアドレスが同一のフレームのアドレスからのオフセットを使用してアクセス可能になるようにします。この結果、このセグメントがソースファイル内で全体として定義されている場合と同一のものが生成されます。

LINK は、それぞれのセグメントの位置合わせのタイプを保存します。これは、セグメントが大規模なセグメントに属していても、セグメント内のコードおよびデータはもとの位置合わせを保存するという意味です。結合されたセグメントが64K バイトより大きい場合、エラーメッセージを表示します。

セグメントのタイプが stack または memory である場合、LINK は public セグメントと同様の結合操作を行います。1つ違うのは、stack の場合、最初のスタックポインタ値を実行可能ファイルにコピーすることということです。このスタック

ポインタ値は､検出された先頭のスタックセグメント(または結合されたスタックセグメント)の終わりまでのオフセットです。

セグメントのタイプがcommonの場合、同一の名前が付けられ、同一のクラスに属するすべてのセグメントと結合します。commonセグメントを結合するとき、各セグメントの開始点を同一のアドレスにロードし、オーバラップしたセグメントを作成します。この結果、結合された最大のセグメントと同一の大きさの単一のセグメントが生成されます。

ソースファイル内で、セグメントのために明示の結合タイプが定義されていない場合、このセグメントのタイプはprivateになります。privateセグメントは結合しません。

## ■ グループ

グループによって、連続しておらず、同一のクラスに属していないセグメントは、同一のフレームアドレスからの相対アドレスでアドレス指定することができます。

LINKは、グループを検出したとき、このグループ内の項目に対するメモリ参照が、同一のフレームアドレスからの相対アドレスになるように調整します。

グループ内のセグメントは、連続している必要はなく、同一のクラスに属する必要もなく、結合タイプが同一である必要もありません。グループ内のすべてのセグメントが、64Kバイト内に収まることだけが必要とされます。

グループは、セグメントがロードされる順序に影響を及ぼしません。クラス名を使用せず、オブジェクトファイルを正しい順序で入力しなかった場合、セグメントが連続しているという保障はありません。実際にLINKは、同一の64Kバイトメモリ中にこのグループに属さないセグメントをロードする可能性があります。LINKは、グループ内のすべてのセグメントが64Kバイト以内かどうかチェックしないので、これが満たされない場合、フィックスアップオーバフローエラーを検出する可能性があります。

## ■ 参照の解決

プログラム中の各セグメントの開始アドレスとすべてのセグメントの組み合わせおよびグループを確立すると、LINKは、ラベルおよび変数の参照を解決します。参照を解決するために、LINKは該当するオフセットおよびセグメントのアドレスを計算し、アセンブラによって生成された一時的な値を新規の値で置き換えます。

LINKは、以下の4つの異なった参照を解決します。

- Short
- Near Self-Relative
- Near Segment-Relative
- Long

計算する値の大きさは、参照のタイプによります。LINKは、参照でエラーを検出した場合、メッセージを表示します。プログラムが16ビットのオフセットを使用して、異なったフレームアドレスを持つセグメント内の命令にアクセスしようとした場合などに、このエラーが発生する可能性があります。

short参照は、参照点から128バイト以内のところになければなりません。LINKはこの参照のために、符号付の8ビットの数を計算します。

near self-relative参照は、同一のセグメントやグループからの相対アドレスにあるデータにアクセスするときに行われます。LINKは、この参照のために、16ビットオフセットを計算します。

near segment-relative参照は、指定されたセグメントやグループ内のデータ、または指定されたセグメントレジスタからの相対アドレスにあるデータにアクセスするときに行われます。LINKは、この参照のために、16ビットオフセットを計算します。

long参照は、CALL命令が他のセグメントまたはグループ内の命令にアクセスするときに行われます。LINKは、この参照のために、16ビットフレームアドレスおよび16ビットオフセットを計算します。

## ■ローディングの順序の制御

空のセグメント定義が示されているダミープログラムを作成し、アセンブルすることでセグメントのローディングの順序を制御することができます。LINKを実行するときには、このファイルを先頭のオブジェクトファイルとして指定します。

つぎのダミープログラムは、CODE、DATA、STACK、CONST、MEMORYという名前のセグメントによって構成されるプログラム中のセグメントのローディングの順序を定義します。

```
CODE      segment   'CODE'
CODE      ends
CONST     segment   'CONST'
CONST     ends
DATA      segment   'DATA'
DATA      ends
STACK     segment  stack   'STACK'
STACK     ends
MEMORY    segment   'MEMORY'
MEMORY    ends
```

このダミープログラムには、プログラム中で使用すべきすべてのクラスのための定義が入っていなければなりません。入っていない場合、ユーザーの希望する順序と一致しない場合があります。

**注意** C、Pascal などの高級言語のプログラムと一緒に、ダミープログラムを使用しないでください。このような言語は、それ自身のローディング順序を定義するからです。

ダミープログラムの終わりに空の MEMORY セグメントを置くと、MEMORY セグメントをプログラム中の最終セグメントとしてロードするように LINK に強制することができます。空のセグメントは、つぎのように指定します。

```
<セグメント名>  segment  memory  '<クラス名>'
<セグメント名>  ends
```

## 2.6 メッセージ

リンカ：LINK の表示するメッセージをアルファベット順に解説します。

About to generate .EXE file
Change diskette in drive<d：> and press<ENTER>

EXE ファイルを作成する処理を中断しています。EXE ファイルを収めるディスクをドライブ<d：>にセット(交換)して、リターンキーを押してください。これは、/P スイッチを指定したときに表示されるメッセージです。

Ambiguous switch error："x"

スイッチ名を省略しすぎて、一意に確定できません。スイッチの最小の省略形を確認してください。たとえば、つぎのようなスイッチの指定をすると、このメッセージが表示され、LINK は処理を中止します。

```
A>LINK /N main ;
```

Array element size mismatch

コモンなアレイ領域が、2 ケ所以上で、異なるサイズのエレメント(文字型と実数型など)で宣言されています(現バージョンの MASM では、コモンなアレイはサポートされていません)。

Attempt to put segment<name> in more than one group in file
<filename>

<filename>中のセグメント<name>は、2 つのグループで重複して定義されています。ソースファイルを修正して、オブジェクトファイルを作り直してください。

Attempt to access data outside segment bounds

セグメント外のデータをアクセスしようとしました。オブジェクトのソースを修正してください。

Bad value for cparMaxAlloc

/C：<n>スイッチで指定した数値が1～65535の範囲外です。

Cannot find library：<filename>.lib. Enter new file spec：

指定されたライブラリファイル<filename>.libが見つかりません。ドライブ名、パス名、ファイル名を確認して、正しいファイル名を入力してください。

Cannot find file

ファイルが見つかりません。正しいファイル名を入力してください。

Cannot nest response files

応答ファイル中に、さらに応答ファイル名が含まれています。応答ファイルはネスト(入れ子)にできません。応答ファイルを修正してください。

Cannot open list file

ディスクまたはディレクトリが一杯で、マップファイルを作成できません。ディスクまたはディレクトリの空きスペースを作ってください。

Cannot open response file

応答ファイルをオープンできません。ファイル名を間違って入力していないかどうか確認してください。

Cannot open run file

ディスクまたはディレクトリが一杯で、実行可能ファイルを作成できません。ディスクまたはディレクトリの空きスペースを作ってください。

Cannot open temporary file

ディスクまはたディレクトリが一杯で、一時作業ファイル(VM.TMP)を作成できません。ディスクまたはディレクトリの空きスペースを作ってください。

Cannot reopen list file

LINKを起動したディスクが交換されたために、マップファイルを再オープンできませんでした。LINKを再起動してください。

Common area longer than 65535 bytes

プログラムが、64Kバイト以上のコモン変数を持っています。

Data record too large

オブジェクトモジュール中のLEDATAレコードに、1024バイト以上のデータが入っています。

Dup record too large

オブジェクトモジュール中のLIDATAレコードに、512バイト以上のデータが入っています。これは構造定義が複雑すぎたり、DUPステートメントのネストが深すぎる(例：ARRAY db 10 dup(11dup(12dup(13dup(…))))ことが原因と考えられます。ソースファイルを修正してください。

"<filename>"is NOT a valid library

指定したライブラリファイル<filename>に問題があります。

Fixup overflow near<num>in segment<name>in<filename>(name)offset <num>

このメッセージが出力される原因としては、つぎのような状況が考えられます。

1) 1グループが64Kバイト以上ある。
2) プログラム中に、セグメントにまたがるshortジャンプ／コールが含まれている。
3) リンクするライブラリ中のルーチンと、同一のデータ項目名が使われている。
4) セグメントの本体中で、EXTRNが用いられている。この例をつぎに示す。

```
CODE    segment public  'code'
extrn   main：far
start   proc    far
call    main
ret
start   endp
CODE    ends
```

つぎのプログラムは前例を修正したもの。

```
extrn   main：far
CODE    segment public  'code'
start   proc    far
call    main
ret
start   endp
CODE    ends
```

Incorrect DOS version

DOSのバージョンが違います。動作可能なDOSのバージョンで起動してください。

Insufficient stack space

メモリ不足で、LINKを実行できません。

Interrupt number exceeds 255

/O：<n>スイッチで、指定された数値が0〜255以外です。

Invalid numeric switch specification

スイッチに、無効なパラメータが指定されています。数値を指定するべきところで文字を指定している(0とOの間違い)、なども原因として考えられます。

Invalid object module

オブジェクトモジュールに問題があります。アセンブルまたはコンパイルしなおしてください。

NEAR/HUGE conflict

コモン変数のnearとhugeの定義が重複しています(現バージョンのMASMでは、コモン変数はサポートされていません)。

Nested left parentheses
Nested right parentheses

オーバーレイを指定する、LINKのコマンドラインに、文法的な誤りがあります。

No object modules specified

オブジェクトファイルが指定されていません。

Object not found

オブジェクトが見つかりません。ファイル中のオブジェクトモジュールを確認してください。

Out of space on list file

マップファイルを書き出し中、ディスクが一杯になって入りません。ディスクの空きスペースをふやして、LINKを再起動してください。

Out of space on run file

.EXE(実行可能)ファイルを書き出し中、ディスクが一杯になって入りません。ディスクの空きスペースをふやして、LINKを再起動してください。

Please replace original diskette in drive <d：> and press <ENTER>

LINKを起動した(オリジナル)ディスクをドライブ<d：>にセットして、リターンキーを押してください。これは/Pスイッチを使用したときに表示されるメッセージで、EXEファイルを書き出し終わると出力されます。

Program entry point at <nnnn：nnnn>

プログラムのエントリポイントのアドレスを、〝セグメント：オフセット〞の形式で表示します。

Relocation table overfolw

16384バイトを超すlongコール/ジャンプ/ポインタが使われています。long参照をshort参照に換えて再処理してください。

Segment limit set too high

/SE：<n>スイッチで指定した数値が大きすぎます。

Segment limit too high

/SE：<n>スイッチで指定した値(または初期設定値：128)で、アロケートテーブルを作るには、メモリが不足しています。値を小さくするか、メモリのフリーエリアを広げてください。

Segment size exceeds 64K

セグメントサイズの制限、64Kバイトを超えています。たとえば、スモールメモリモデルでありながら64Kバイト以上のコード、コンパクトメモリモデルでありながら64Kバイト以上のデータを持っているなどが相当します。メモリモデルを変更して、コンパイルとリンク処理を行ってください。

Stack size exceeds 65536 bytes

/STACKスイッチで指定した値が、65536バイトを超えています。

Symbol already defined

シンボル名はすでに定義されています。名前を変更して再処理してください。

Symbol defined more than once

シンボル名が多重定義されています。名前を変更して再処理してください。

Symbol table overflow

プログラムが、256Kバイト以上のシンボル情報(public、extrn、segment、group、class、fileなど)を含んでいます。セグメントやモジュールの結合などを行って、オブジェクトファイルを作り直してください。

Terminated by user

[CTRL]＋[C]キー入力によって、リンク処理が中止されました。

Too many external symbols in one module

オブジェクトモジュール中の外部シンボルが多過ぎます。モジュールを分割するなどしてください。

Too many group-, segment-, and class-names in one module

グループ名、セグメント名、クラス名が多過ぎます。数を減らしてオブジェクトファイルを作り直してください。

Too many groups

プログラム中で10個以上のグループを定義しています。

Too many GRPDEFs in one module

1モジュール中に10個以上のGRPDEFがあります。数を減らすか、モジュールを分割してください。

Too many libraries

17個以上のライブラリファイルを指定してます。16個以下にしてください。

Too many overlays

64個以上のオーバーレイを定義しています。数を減らしてください。

Too many segments

プログラム中のセグメントが多すぎます。/SEGMENTスイッチのパラメータを適当な数値にふやして、再びリンク処理を行ってください。

Too many segments in one module

オブジェクトモジュールに、256個以上のセグメントが含まれています。オブジェクトを分割するか、セグメントを連結して数を減らしてください。

Too many TYPDEFs

オブジェクトモジュール中のTYPDEFレコードが多過ぎます。このレコードは、コンパイラがコモン変数の領域のために生成したものです(現バージョンのMASMは、コモン変数をサポートしていません)。

Unexpected end-of-file on library

ライブラリを収めたディスクが、外された可能性が高いと考えられます。ディスクを確認してください。

Unexpected end-of-file on scratch file VM.TMP

VM.TMPファイルの作られていたディスクが、外された可能性が高いと考えられます。ディスクを確認して、LINKを再起動してください。

Unmatched left parenthesis

オーバーレイを指定するLINKのコマンドラインに、文法エラーがあります。

Unmatched right parenthesis

オーバーレイを指定するLINKのコマンドラインに、文法エラーがあります。

Unrecognized switch error："＜switch＞"

スイッチとして、無効な文字が指定されています。

VM.TMP is an illegal file name and has been ignored

オブジェクトファイル名に“VM.TMP”が使われています。ファイル名を変更してください。

Warning : no stack segment

プログラムは、スタックスペースセグメントを割り当てるステートメントを含んでいません。

Warning : too many public symbols

パブリックシンボルのソート(並べ換え)マップを要求しましたが、シンボル数が多過ぎて、ソートできませんでした。LINK は、ソートしていないパブリックシンボルのマップを作成しました。

# 第3章

# MAPSYM：シンボルマップユーティリティ



## 3.1 イントロダクション

シンボルマップユーティリティ：MAPSYMはLINKが作成したマップファイルを利用して、SYMDEB用のシンボルファイルを作成します。

## 3.2 シンボルファイルの作成

SYMDEB用のシンボルファイルを作るために、まずLINKでマップファイルを作成します。このとき、LINKのスイッチとして/M(/MAP)と/LI(/LINENUMBER)の2つを指定して、マップファイルを作成します(詳しくは、第2章「LINK：リンカ」を参照してください)。ここで作成されたマップファイルからMAPSYMでシンボルファイルを作成します。

MAPSYMコマンドの書式はつぎのとおりです。

```
MAPSYM [-L | /L] <ファイル名>
```

<ファイル名>には、LINKで作成されたマップファイル(拡張子は.MAP)の名前を指定します。ドライブ名やパス名を付けて指定することもできます。拡張子を省略した場合、".MAP"と解釈されます。作成されたシンボルファイルの拡張子は".SYM"になります。このシンボルファイルは、カレントディレクトリに作成されます。

スイッチの-Lと/Lは同じもので、プログラムに定義されたグループ名、プログラムの開始および行番号の有無などの変換に関する情報を表示します。

たとえば、マップファイル"file.map"からシンボルファイル"file.sym"を作成する場合は、つぎのように入力します。

```
MAPSYM file.map
```

## 3.3　メッセージ

MAPSYM が表示するメッセージをアルファベット順に解説します。

Can't create：<filename>

カレントのドライブまたはディレクトリが一杯です。空きスペースを作ってください。

Can't open MAP file：<filename>

指定されたマップファイルがありません。ファイル名を確認してください。

mapsym：out of memory

処理に必要なメモリが足りません。

mapsym：segment table(n)exceeded.

マップファイル中のセグメント数が 1024 個を超えています。(n)はマップファイル中の総セグメント数を示しています。

No entry point, assume 0000：0100

プログラムのエントリポイントがないため、0000：0100 番地を想定します。

No public symbols
Re-link file with /m switch!

LINK で/M スイッチを指定しなかったために、パブリックシンボル名がありません。もう一度リンクしなおしてください。

Program entry point at <segment>：<offset>

プログラムのエントリポイントのアドレスを表示します。

Unexpected eof reading：<filename>

マップファイルが正しく生成されていないので、もう一度リンクしなおしてください。

usage：MAPSYM [/l] maplist

コマンドラインに誤りがあります。正しく指定してください。

Write fail on：<filename>

シンボルファイルを作成中にエラーが発生しました。

# 第4章

# SYMDEB：シンボリックデバッガ



## 4.1 イントロダクション

シンボリックデバッガ：SYMDEBは、メモリやレジスタの操作、プログラムのトレースや逆アセンブルといった一般的なデバッガの機能に加えて、サブルーチンや変数などをシンボル(名前)で参照することのできるシンボリックデバッガです。また、SYMDEBの強力な機能として、ソースの表示と番号付けがあります。この機能によって、高級言語のデバッグが機械語のレベルだけでなく、ソースのレベルでも行えます。

このような高度なデバッグを行うためには、シンボルファイルを作成しておく必要があります。シンボルファイルの作成については、第3章「MAPSYM：シンボルマップユーティリティ」を参照してください。

## 4.2 SYMDEBの起動

SYMDEBを起動すると、SYMDEBのプロンプト(-)が表示され、SYMDEBのコマンドを入力できる状態になります。SYMDEBは、つぎのような書式で起動します。

SYMDEB [<シンボルファイル名>] [<実行ファイル名>] [<引数リスト>]

<シンボルファイル名>

MAPSYMで作成されたシンボルファイル(拡張子".SYM")のファイル名を指定します。複数のファイルを指定することもでき、この場合、次の<実行ファイル名>の前にすべてのシンボルファイルを指定します。

<実行ファイル名>

デバッグする実行可能ファイルのファイル名を指定します。

<引数リスト>

<実行ファイル>に与える引数を記述します。

**例：**　SYMDEB file.com

この例のように実行ファイルのみ指定した場合、SYMDEBは、使用可能なメモリの最下位セグメントの先頭から100H(256)バイトのプログラム用のヘッダを作成し、100H番地以降にファイルをロードします。ただし、EXE形式のファイルの場合は、ファイルのヘッダ部分で定義されたアドレスにロードします。

さらに、ファイルの大きさ(バイト単位)をBX：CXレジスタにセットし、セグメントと他のレジスタをファイルのヘッダ部分で定義された個々の値にセットします。

**例：**　SYMDEB file.sym file.exe

この例では、"file.sym"からシンボル情報を読み込みます。次に、プログラム用のヘッダを作成した後に、"file.exe"をロードします。

**例：**　symdeb file.sym file.exe test.dat/m/b

この例では、"test.dat/m/b"をプログラム用のヘッダにコピーし、それから"file.exe"をロードします。"test.dat/m/b"は"file.exe"の引数になります。

すべてのパラメータを省略して、単に"SYMDEB"と入力して起動することもできます。起動後、SYMDEBのプロンプトが表示され、SYMDEBのコマンドが入力できます。

**例：**

```
A>symdeb
Microsoft Symbolic Debug Utility
Version 3.10
(C)Copyright Microsoft Corp 1984,1985
Processor is [80286]
-
```

## 4.3　コントロールキーの使用

SYMDEBを実行中、誤りの修正やコマンドの停止には、MS-DOSユーザーズリファレンスマニュアルで述べられているコントロールキャラクタやテンプレート機能を用いることができます。

SYMDEBは CTRL ＋ C キーを押すと、実行中のコマンドを停止してプロンプトを表示します。CTRL ＋ S キーを押すと、SYMDEBはコマンド出力を一時的に中断します。ただし、Redirectionコマンドで"AUX"を利用しているときには、CTRL ＋ C キーによる停止と CTRL ＋ S キーによる中断は無効ですので注意してください。

また、Goコマンドでプログラムを実行中、CTRL ＋ C キーで停止することもできますが、これはプログラムが入出力の処理をしているときに限られます。

# 4.4 コマンドとパラメータの表記

SYMDEB のコマンドをつぎの表に示します。

| コマンド名 | 書式 | 機　能 |
|---|---|---|
| Assemble | A | アセンブル |
| Breakpoint Set | BP | ブレークポイントの設定 |
| Breakpoint Clear | BC | ブレークポイントの解除 |
| Breakpoint Disable | BD | ブレークポイントの禁止 |
| Breakpoint Enable | BE | ブレークポイントの許可 |
| Breakpoint List | BL | ブレークポイントのリスト |
| Comment | * | コメントの表示 |
| Compare | C | 比較 |
| Display | ? | 式の値の表示 |
| Dump Ascii | DA | ASCII ダンプ |
| Dump Bytes | DB | バイトダンプ |
| Dump Words | DW | ワードダンプ |
| Dump Doublewords | DD | ダブルワードダンプ |
| Dump Short Reals | DS | ショート実数のダンプ |
| Dump Long Reals | DL | ロング実数のダンプ |
| Dump Ten-byte Reals | DT | 10 バイト実数のダンプ |
| Dump | D | ダンプ |
| Enter Ascii | EA | ASCII コードの入力 |
| Enter Bytes | EB | バイトデータの入力 |
| Enter Words | EW | ワードデータの入力 |
| Enter Doublewords | ED | ダブルワードデータの入力 |
| Enter Short Reals | ES | ショート実数の入力 |
| Enter Long Reals | EL | ロング実数の入力 |
| Enter Ten-byte Reals | ET | 10 バイト実数の入力 |
| Enter | E | データの入力 |
| Examine Symbol Map | X | シンボル情報の表示 |
| Fill | F | フィル |
| Go | G | 実行 |
| Help | ? | コマンド一覧の表示 |
| Hex | H | 16 進計算 |
| Input | I | ポート入力 |
| Load | L | ロード |
| Move | M | 移動 |
| Name | N | 名前のセット |
| Open Map | XO | シンボルマップ、セグメントの設定 |
| Output | O | ポート出力 |
| P Trace | P | 割り込みに対応したトレース |
| Quit | Q | SYMDEB の終了 |
| Redirection | < | 入力の切り換え |
| | > | 出力の切り換え |
| | = | 入力と出力の切り換え |
| Register | R | レジスタの表示と設定 |
| Search | S | サーチ |
| Set Source Mode | S | ソース形式のセット |
| Shell Escape | ! | MS-DOS コマンドの実行 |
| Source Line | . | 現在のソースラインの表示 |
| Stack Trace | K | スタックフレームの表示 |

| | | |
|---|---|---|
| Symbol Set | Z | シンボリックアドレスに値をセット |
| Trace | T | トレース |
| Unassemble | U | 逆アセンブル |
| View | V | ソースラインの表示 |
| Write | W | 書き込み |

コマンドの後には、パラメータを必要とするものもあります。複数のパラメータを指定する場合は、カンマ( , )またはスペースでパラメータを区切ります。コマンド名とパラメータについては、大文字と小文字の区別は行われません。

以後の項では、コマンドのパラメータについて詳しく説明します。

## ■ シンボル

シンボルは、レジスタ、絶対値、セグメントアドレス、セグメントのオフセットを表す名前です。シンボルは、必ず文字または記号( _、?、@、$)で始まります。

レジスタを表すシンボル(レジスタ名)は常に使用可能ですが、他のシンボルはシンボルファイルがロードされている場合にのみ使用することができます。レジスタ名は、他のシンボルに優先しますから、レジスタ名と同じ名前のシンボルは無視されます。

例：
```
AX
main
DGROUP
IP
```

## ■ 数値

SYMDEB では、2、8、10、16 進数を用いることができ、添え字でその基数を表します。添え字は、つぎのとおりです。

| 添え字 | タイプ | 使用できる数字 |
|---|---|---|
| Y | 2 進数 | 01 |
| O, Q | 8 進数 | 01234567 |
| T | 10 進数 | 0123456789 |
| H | 16 進数 | 0123456789ABCDEF |

デフォルトは 16 進数なので、添え字を省略すると 16 進数と見なされます。

数には、数字をいくつでも入れることができますが、SYMDEB では、数が 65535 より大きい場合は、先頭の数字を切ります。また、先頭にゼロがあっても無視されます。

例：
```
0111111Y    77Q    63T    3FH    3F
01001010100101Y    112450    4773T    12AH    12A5
```

## ■ アドレス

アドレスは、"<セグメント>:<オフセット>"という形式で指定します。セグメントとオフセットは、数値やレジスタ名またはシンボルで指定します。セグメントを省略すると、デフォルトのセグメントアドレスが与えられます。デフォルトのセグメントは、Assemble コマンド、Go コマンド、Load コマンド、PTrace コマンド、Unassemble コマンド、Write コマンドの場合は CS レジスタ、それ以外のコマンドの場合は DS レジスタになります。

例： cs：0100
04BA：IP
CS：main
DGROUP：count

## ■ アドレスのレンジ指定

アドレスのレンジ(範囲)を指定するには、つぎのような 2 種類の方法があります。

書式 1： 開始アドレスと終了アドレスを指定する方法

<開始アドレス> <終了アドレス>

この書式では、<開始アドレス>と<終了アドレス>をスペースで区切って指定します。<終了アドレス>を省略した場合、128 バイトの範囲になります。<終了アドレス>にはセグメントの指定はできず、この場合そのセグメントは〈開始アドレス>で指定したものとおなじになります。

例： cs：100 110
_main _main+20

書式 2： 開始アドレスと長さを指定する方法

<開始アドレス> L<数値>

この書式では、コマンドで処理する範囲は、<開始アドレス>から<数値>で示される範囲までとなります。この書式が使えるのは、Dump コマンド、Fill コマンド、Search コマンド、Unassemble コマンドです。対象範囲の大きさや種類はコマンドによって異なります。たとえば、Dump Bytes コマンドはバイト単位、Dump Words コマンドはワード単位、Unassemble コマンドは命令単位になります。

例： DGROUP：table L100
main L20

SYMDEB

## ■ 行番号

行番号を指定するときの書式は、つぎのようになっています。

**.±行番号**
**.[ファイル名：]行番号**
**.シンボル[±行番号]**

ソースプログラム中の行番号は、先頭にピリオド(.)を付けて、10進数で指定します。

最初の書式は、相対行番号を指定するものです。番号は、現在のソース行から新しい行までのオフセット(行単位)です。プラス記号(+)の場合ソースファイルの後ろ、マイナス記号(-)の場合はソースファイルの前を指定することになります。

2番目の書式は、絶対行番号を指定するものです。ファイル名を指定すると、これによって示されたシンボルファイルに対応するソースファイル内の行番号を指定したことになります。ファイル名を指定しないと、現在のファイルの行番号を指定したことになります。

3番目の書式は、シンボルからのオフセット(行番号)を指定するものです。プラス記号(+)の場合ソースファイルの後ろ、マイナス記号(-)の場合はソースファイルの前を指定することになります。

| | | |
|---|---|---|
| 例： | .+5 | ；現在行から5行下の行 |
| | .10 | ；現ソースファイルの10行目 |
| | .sample：10 | ；〝sample〞というシンボルファイルが指定する<br>ソースファイルの10行目 |
| | .main | ；〝main〞の1行目 |
| | .main+5 | ；〝main〞の5行目 |

シンボル〝main〞は、アドレスの指定にも使用できます。ただし、〝main+3〞が〝main〞から3バイト目のアドレスを指定するのに対して、〝.main+3〞は、〝main〞から3行目のソース行を指定するので注意してください。

## ■ ストリング

ストリングは、クォーテーション(”または’)で囲まれた任意の長さの文字列です。ストリングには、シングルクォーテーション(’)かダブルクォーテーション(”)を含めることもできます。ただし、始まりのクォーテーションと終わりのクォーテーションは同じものでなくてはなりません。ストリング中にさらにクォーテーションを入れるときは、そのクォーテーションを2個つづけて入力するか、別の種類のクォーテーションを用います。

例： 'This is a string.'
''This is a string.''
'This ''string'' is okay.'
''This'' ''string'' ''is okay.''
'This ''string'' is okay.'
''This 'string' is okay.''

## ■式

式は、8ビット、16ビット、32ビットの値を求めるオペランドと演算子の組み合わせです。式は、すべてのコマンドで、数値として用いることができます。

式は、すべてのシンボルと数字、アドレスを、つぎに示す単項演算子および2項演算子を用いて組み合わせることができます。

SYMDEB

●単項演算子

| 演算子 | 意　味 | 優先順位 |
|---|---|---|
| ＋ | 単項プラス | 高 |
| － | 単項マイナス | ↑ |
| NOT | 1の補数 | |
| SEG | オペランドのセグメント | |
| OFF | オペランドのオフセット | |
| BY | 特定アドレスからのバイト | |
| WO | 特定アドレスからのワード | |
| DW | 特定アドレスからのダブルワード | |
| POI | 特定アドレスからのポインタ(4バイト) | |
| PORT | 特定ポートからの1バイト | ↓ |
| WPORT | 特定ポートからの1ワード | 低 |

●2項演算子

| 演算子 | 意　味 | 優先順位 |
|---|---|---|
| ＊ | 乗算 | 高 |
| / | 整数の除算 | ↑ |
| MOD | 余り | |
| : | セグメントのオーバライド | |
| ＋ | 加算 | |
| － | 減算 | |
| AND | 論理積 | |
| XOR | 排他的論理和 | ↓ |
| OR | 論理和 | 低 |

式は、演算子の優先順に実行されます。隣りあった演算子の優先順位が同じ場合、式は左から右へ評価されます。カッコを用いれば、この順序を変えることができます。

例：

| 式 | 式の評価 |
|---|---|
| 4+2＊3 | ；= 10(0AH) |
| SEG 0001：0002 | ；= 1 |
| OFF 0001：0002 | ；= 2 |
| 4+(2＊3) | ；= 10(0AH) |
| (4+2)＊3 | ；= 18(12H) |

## 4.5　SYMDEB のコマンド

本節では、SYMDEB のコマンドをアルファベット順に解説します。SYMDEB のコマンドには、パラメータを必要とするものがありますが、パラメータを省略したときは、デフォルトの値や、直前に行ったコマンドの結果のレジスタ値が用いられます。

# A Assemble

## 命令をアセンブルしてメモリに入れる

**書　式**　　A [<アドレス>]

### 解　説

8086/8087/8088/80186/80286/80287 の命令をアセンブルして、その命令コードを指定されたアドレスに入れます(80286 のプロテクトモードは除く)。コマンドは、アドレスと指定されたアドレスの命令コードを表示して、次の命令のためのプロンプトを出します。アドレスを入力しないと、CS レジスタと IP レジスタの現在値によって与えられるアドレスから開始されます。アセンブルを終了したいときは、リターンキーのみ押します。

命令の入力に関する規則はつぎのとおりです。

1. WAIT や REP などのプリフィックス命令は、それを適用する命令の前に指定します。これらは、別の行に分けて入力することもできます。

2. FAR リターンのための命令は、RETF です。

3. ストリング(文字列)の操作に関する命令は、ストリングサイズを明確に指定します。たとえば、ワード列を移動する場合には MOVSW を用い、バイト列を移動する場合は、MOVSB を使用します。

4. JMP や CALL 命令では、SHORT、NEAR、FAR が自動的に決定されます。これらは、NEAR または FAR プリフィックスを用いれば無効にすることができます。なお、NEAR プリフィックスは、NE と略すことができます。

5. オペランドがワード単位かバイト単位か判断できない場合は、WORD PTR または BYTE PTR プリフィックスを用いて、データの種類を明確にしなくてはなりません。WORD PTR は WO、BYTE PTR は BY と略すことができます。

例：
```
MOV    WORD    PTR    [BP],1
MOV    BYTE    PTR    [SI-1],SYMBOL
```

SYMDEB

6. オペランドがアドレスを示す場合は、〝[ ]〞でくくります。そうでない場合は、値そのものを示すオペランドと見なされます。

**例：**

```
MOV    AX,21
MOV    AX,[21]
```

7. 疑似命令の DB と DW を使うことができます。DB はバイト値を、DW はワード値をメモリに直接書き込みます。

**例：**

```
DB     1, 2, 3, 4, "THIS IS AN EXAMPLE"
DB     'THIS IS A QUOTE : "'
DB     "THIS IS A QUOTE : '"
DW     1000, 2000, 3000, "BACH"
```

8. レジスタ間接指定が利用できます。

**例：**

```
ADD    BX,34[BP+2][SI-1]
POP    [BP+DI]
PUSH   [SI]
```

9. すべての同義命令コード（LOOPZ と LOOPE、JA と JNBE など）の表記が使用できます。

10. 8087 命令コードは、明確に指定しなくてはいけません。たとえば、WAIT は 8086 命令コード、FWAIT は 8087 命令コードです。また、8087 命令コードを使用するときは、システムに 8087 数値演算プロセッサが装着されている必要があります。

**サンプル**

```
-A CS:_main
```

この例では、〝CS：_main〞によって与えられるアドレスからアセンブルが開始されます。

```
-A 04BA:0100
```

この例では、〝04BA：0100〞によって与えられるアドレスからアセンブルが開始されます。

# BP

## Breakpoint Set

## ブレークポイントを設定する

**書　式**　BP[n] <アドレス> [<回数>]

### 解　説

指定されたアドレスにブレークポイントを設定します。プログラムの実行がブレークポイントに達すると、プログラムは停止して、レジスタとフラグの現在値を表示します。このブレークポイントは、Gコマンドが作成するブレークポイントと違って、BCコマンドを用いて無効にしない限り、プログラム内にとどまります。

ブレークポイントは、10個まで設定できます。[n]には、ブレークポイントの番号(0〜9)を指定します。BPと[n]の間にスペースを入れてはいけません。nを省略した場合は、最初に使用可能なブレークポイント番号が割り当てられます。

＜アドレス＞には、ブレークポイントのアドレスを指定しますが、これは命令コードの最初のバイトでなければなりません。

＜回数＞には、ブレークポイントが有効になるまでに無視される回数を指定します。

**サンプル**

```
-BP _main
```

この例では、ブレークポイントが"_main"に設定されます。

```
-BP8 add
```

この例では、ブレークポイント8番が"add"に設定されます。

```
-BP 100 10
```

この例では、ブレークポイントは、現在のCSセグメントの100Hに作成されます。このブレークポイントは、有効になるまでに16(10H)回無視されます。

SYMDEB

# BC

## Breakpoint Clear

## ブレークポイントを解除する

**書　式**　　BC {<リスト> | *}

### 解　説

BP コマンドで設定されたブレークポイントを削除します。<リスト>を入力すると、そこに挙げられているブレークポイントを複数削除します。リストは、0 から 9 の整数値の組み合わせです。アスタリスク(*)を指定すると、すべてのブレークポイントを取り除きます。

**サンプル**

```
-BC 0 4 8
```

この例では、ブレークポイント 0、4、8 番が取り除かれます。

```
-BC *
```

この例では、すべてのブレークポイントが取り除かれます。

# BD

## Breakpoint Disable

## ブレークポイントを一時的に無効にする

**書　式**　BD {<リスト> | *}

### 解　説

BD コマンドは、一時的にブレークポイントを無効にします。これは BE コマンドを用いていつでも復元できます。

<リスト>を入力すると、コマンドは、そこで挙げられている番号のブレークポイントを無効にします。<リスト>の内容は、0 から 9 の整数値の組み合わせです。アスタリスク(*)を入力すると、すべてのブレークポイントを無効にします。

**サンプル**

```
-BD 0 4 8
```

この例では、ブレークポイント 0、4、8 番が無効になります。

```
-BD *
```

この例では、すべてのブレークポイントが無効になります。

# BE

## Breakpoint Enable

### ブレークポイントを有効にする

**書　式**　BE {<リスト> | *}

## 解　説

BD コマンドによって一時的に無効となったブレークポイントを有効(復元)にします。<リスト>を入力すると、そこに挙げられている複数のブレークポイントを有効にします。リストは、0から9の整数値の組み合わせです。アスタリスク(*)を入力すると、すべてのブレークポイントを有効にします。

**サンプル**

```
-BE 0 4 8
```

この例では、ブレークポイント0、4、8番を有効にします。

```
-BE *
```

この例では、すべてのブレークポイントを有効にします。

# BL Breakpoint List

## ブレークポイントの情報を表示する

**書式** BL

### 解説

BPコマンドで設定したブレークポイントの情報を表示します。実行すると、ブレークポイント番号、有効/無効の状態、ブレークポイントのアドレス、<回数>の残り数、<回数>の設定値(カッコ内)を表示します。

ブレークポイントが設定されていなければ、何も表示されません。

**サンプル**

```
-BL
0  e  04BA:0100
4  d  04BA:0503  4(10)
8  e  0D2D:0001  3(3)
```

この例では、ブレークポイント0番と8番が有効(e)で、ブレークポイント4番は無効(d)になっています。ブレークポイント4番の<回数>の設定は10で、あと4回の残り回数があります。ブレークポイント8番の最初のパス数は3で、その3つの<回数>をすべて残しています。ブレークポイント0番には、最初の<回数>がありませんが、これは1にセットされたということです。

SYMDEB



# Comment

## 注釈文(コメント)を表示する

**書　式**　　* <コメント>

### 解　　説

アスタリスク(*)につづいて入力した文字を標準出力にエコーバックします。このコマンドは、Redirection(入出力の切り換え)コマンドで標準出力をファイルやプリンタに切り換えているときに使用し、デバッグ作業の流れ(経過・工程)を記録として残しておきたいときなどに役立ちます。

**サンプル**

```
-R CX 80
-* Change the count in CX to 80
Change the count in CX to 80
```

# C Compare

## メモリ内容を比較する

**書式** C <レンジ> <アドレス>

### 解説

＜レンジ＞で指定した範囲のメモリの内容と、＜アドレス＞から始まるメモリの内容を比較します。その結果、内容がすべて一致したときは、何も表示しません。一致しない場合は、その一致しない内容が表示されます。

SYMDEB

**サンプル**

```
-C 100,1FF 300
```

この例では、メモリの100H番地から1FFH番地までのブロックと、300H番地から3FFH番地までのブロックを比較します。

```
-C 100 L 100 300
```

この例では、メモリの100H番地から256(100H)バイト目までと、300H番地から256バイト目まで比較します。

# ?

## Display

## <式>の値を表示する

**書　　式**　　? <式>

### 解　　説

＜式＞を入力すると、それを評価して値を表示します。＜式＞は、数、シンボル、アドレスおよび演算子を組み合わせたものです。コマンドは、式を評価してから、様々な形式でその値を表示します。形式には、完全アドレス16ビット16進数、32ビット16進数、10進数(カッコで囲まれている)、ストリング値(ダブルクォーテーションで囲まれている)があります。

**サンプル**

```
-? 3*4
```

この例では、式"3*4"の値が表示されます。

```
-? DS:table
```

この例では、シンボルアドレス"DS：table"の値が表示されます。

```
-? WO DGROUP:_bufsiz
```

この例では、シンボルアドレス"DGROUP：_bufsiz"のワードが表示されます。

# DA

## Dump Ascii

## メモリ内容を ASCII 文字で表示する

**書　式**　DA [<アドレス> | <レンジ>]

### 解　説

指定した＜アドレス＞または＜レンジ＞のメモリ内容を ASCII 文字で表示します。キャリッジリターンやラインフィードなどの印字できないコードは、ピリオド(.)で表示されます。

＜アドレス＞を指定すると、最初の〝00H″にあたるまで、または 128 バイト分を表示するまで、ASCII 文字を表示し続けます。＜レンジ＞を指定すると、コマンドは、その範囲が終わるまで、ASCII 文字の表示を続けます。

＜アドレス＞も＜レンジ＞も指定しなかった場合、コマンドは最初の〝00H″まで、または 128 バイトを表示するまで、ASCII 文字の表示を行います。この場合、前回に表示された最終アドレスの次のアドレスから、表示が開始されます。

**サンプル**

```
-DA CS : 100 110
04BA : 0100 A STRING..Text..
```

この例では、〝CS : 100″から〝CS : 110″までの ASCII 文字が表示されます。印字できないコードはピリオド(.)で表示されます。

```
-DA name
```

この例では、シンボルアドレス〝name″から ASCII 文字が表示されます。

SYMDEB

# DB

## Dump Bytes

### メモリ内容を16進バイトとASCII文字で表示する

**書　　式**　　DB [<アドレス> | <レンジ>]

## 解　　説

指定した＜アドレス＞から128(100H)バイト分、または＜レンジ＞で指定された範囲のメモリ内容を、16進数とASCII文字で表示します。各行には、16個の16進バイト値とそれに対応するASCII文字が表示されます。印字できないASCII文字は、ピリオド(.)で示されます。

＜アドレス＞も＜レンジ＞も指定しなかった場合、前回のDumpコマンドで表示された最終アドレスの次のアドレスから、表示が開始されます。

**サンプル**

```
-DB CS:100 110
04BA:0100 41 20 73 74 72 69 6E 67-04 01 01 54 65 78 74 0D A string…Text.
04BA:0110 0A
```

この例では、“CS：100”から“110”までのバイト値が表示されます。対応するASCII文字は右側に表示されます。

```
-DB
```

この例では、前回の表示アドレスの次から128バイト分が表示されます。たとえば、前回のDumpコマンドの際の最終バイトが“04BA：0110”だった場合、“04BA：0111”から表示が開始されます。

```
-DB table table+5
```

この例では、シンボルアドレス“table”から始まる6バイトが表示されます。

# DW

## Dump Words

## メモリ内容をワード(2 バイト)単位で表示する

**書　式**　DW [<アドレス> | <レンジ>]

### 解　説

指定した＜アドレス＞から 64 ワード分、または＜レンジ＞で指定された範囲のメモリ内容を、ワード(2 バイト)の 16 進数で表示します。各行には、8 個の 16 進ワード値が表示されます。

＜アドレス＞も＜レンジ＞も指定しなかった場合、前回の Dump コマンドで表示された最終アドレスの次のアドレスから、表示が開始されます。

**サンプル**

```
-DW CS：100 110
04BA：0100 2041 7473 6972 676E 0104 5405 7865 0A0D
04BA：0110 002E
```

この例では、〝CS：100〞から〝CS：110〞までのワード値が表示されます。

```
-DW
```

この例では、現在のダンプアドレスから 64 ワード分が表示されます。たとえば、前回の Dump コマンドの際の最終バイトが〝04BA：0110〞だった場合、〝04BA：0111〞からワードの表示が開始されます。

```
-DW table table+5
```

この例では、シンボルアドレス〝table〞から〝table+5〞までのレンジのワードが表示されます。

# DD

## Dump Doublewords

## メモリ内容をダブルワード(4バイト)単位で表示する

**書　式**　DD [<アドレス> | <レンジ>]

### 解　説

指定した<アドレス>から32個のダブルワード分、または<レンジ>で指定された範囲のメモリ内容を、ダブルワード(4バイト)の16進数で表示します。各行には、4個の16進ダブルワード値が表示されます。

<アドレス>も<レンジ>も指定しなかった場合、前回のDumpコマンドで表示された最終アドレスの次のアドレスから、表示が開始されます。

**サンプル**

```
-DD CS:100 110
04BA:0100 7473:2041 676E:6972 5054:0140 0A0D:7865
04BA:0110 0000:002E
```

この例では、〝CS:100″から〝CS:110″までのダブルワードが表示されます。

```
-DD
```

この例では、現在のダンプアドレスから32個のダブルワードが表示されます。たとえば、前回のDumpコマンドの最終バイトが04BA:0110だった場合、04BA:0111からダブルワードの表示が開始されます。

```
-DD table table+5
```

この例では、シンボルアドレス〝table″から〝table+5″までのレンジのダブルワードが表示されます。

# DS

## Dump Short Reals

## メモリ内容をショート(4 バイト)の浮動小数点数で表示する

**書　式**　　DS　[<アドレス> | <レンジ>]

### 解　　説

指定した<アドレス>または<レンジ>のメモリ内容を、ショート(4 バイト)浮動小数点数(実数)で表示します。16 進数と 10 進数で表示され、10 進数の表示形式は、つぎのとおりです。

±.dd...dE　±mm

SYMDEB は表示を行う前にショート実数をロング実数に変換します。そのため、最大 16 個の 10 進数を表示しますが、最初の 7 個の数字のみが有効です。コマンドは指定したレンジが終わるまで、または最初の 1 個のショート実数を表示するまで値を表示します。

**サンプル**

```
-DS ds：100
04BA：0100 00 00 20 40   +.25E+1
```

この例では、アドレス〝ds：100〟のショート実数が表示されます。1 行につき 1 個の値だけが表示されます。

```
-DS pi
```

この例では、シンボルアドレス〝pi〟のショート実数が表示されます。

SYMDEB

# DL Dump Long Reals

## メモリ内容をロング(8バイト)の浮動小数点数で表示する

**書　　式**　DL [<アドレス> | <レンジ>]

## 解　　説

指定した＜アドレス＞または＜レンジ＞のメモリ内容を、ロング(8バイト)浮動小数点数(実数)で表示します。16進数と10進数で表示され、10進数の表示形式は、つぎのとおりです。

±.dd...dE　±mm

最高16個の10進数が表示されます。

コマンドは指定したレンジが終わるまで、または最初の1個のロング実数を表示するまで値を表示します。

**サンプル**

```
-DL ds:100
04BA:0100 86 37 6B F0 BE 2A 57 3F +.14139993273678774E-2
```

この例では、アドレス〝ds：100″のロング実数が表示されます。1行につき1個の値だけが表示されます。

```
-DL gamma
```

この例では、シンボルアドレス〝gamma″のロング実数が表示されます。

# DT

## Dump Ten-byte Reals

## メモリ内容を10バイトの浮動小数点数で表示する

**書式** DT [<アドレス> | <レンジ>]

### 解説

指定した<アドレス>または<レンジ>のメモリ内容を、10バイトの浮動小数点数(実数)で表示します。16進数と10進数で表示され、10進数の表示形式は、つぎのとおりです。

±.dd...dE ±mm

最高16個の10進数が表示されます。

コマンドは指定したレンジが終わるまで、または最初の1個の実数を表示するまで値を表示します。

**サンプル**

```
-DT  ds:100
04BA:0100 86 37 6B F0 BE 2A 57 3F 00 00 +.14139993273678774E-2
```

この例では、アドレス"ds：100"の10バイト実数が表示されます。1行につき1個の数だけが表示されます。

```
-DT gamma
```

この例では、シンボルアドレス"gamma"の10バイト実数が表示されます。

SYMDEB

# D　Dump

## メモリ領域の内容を表示する

**書　　式**　　D [<アドレス> | <レンジ>]

### 解　　説

指定した＜アドレス＞または＜レンジ＞のメモリ内容を、直前に実行されたDumpコマンドと同じ形式でメモリを表示します。デフォルトでは、DBコマンドと同じ形式です。

**サンプル**

```
-DA ds：100
04BA：0100 A string..
-D
04BA：010B Text...
```

この例では､DAコマンドで表示された直後のアドレスから､ASCII文字を表示します。

```
-DW ds：100 101
04BA：0100 2041
-D ds：324 325
04BA：0324 FE31
```

この例では、Dコマンドはアドレス〝ds：324″のワードを表示します。

```
-DS pi
04BA：0100 00 00 20 40+.25E+1
-D gamma
04BA：0104 00 00 20 40+.25E+1
```

この例では、Dコマンドは、シンボルアドレス〝pi″の実数を表示します。

# EA

## Enter Ascii

## 指定したアドレスにASCIIコード値を入れる

**書式**　EA <アドレス>[<リスト>]

### 解説

<リスト>には、ストリングか16進のバイト値、またはその両方をスペースかカンマで区切って指定します。ストリングを記述するときは、クォーテーション（"または'）で囲みます。<リスト>を指定しないと、アドレスとその現在の値およびピリオド(.)を表示して、値の入力を待ちます。この場合の操作は、EBコマンドとまったく同じです。

**サンプル**

```
-EA data_seg：msg2 "Can't open file"
```

この例では、ストリング〝Can't open file〟のASCIIコードの値が、アドレス〝date_seg：msg2〟以降に書き込まれます。

SYMDEB

# EB

## Enter Bytes

### 指定したアドレスにバイト値を入れる

**書　式**　EB <アドレス> [<リスト>]

## 解　説

＜リスト＞には、ストリングか16進のバイト値、またはその両方をスペースかカンマで区切って指定します。ストリングを記述するときは、クォーテーション(" または ')で囲みます。EBコマンドは、EAコマンドとまったく同じ働きをします。＜リスト＞を指定しないと、アドレスとその現在値およびピリオド(.)を表示して、値の入力を待ちます。ここでは、つぎのように操作します。

1. 新しい値を入力するときは、16進数の1桁か2桁をタイプします。それ以上の余分な数字や文字は無視されます。

2. 次のアドレスへ進むときは、[スペース]キーを押します。

3. 前のアドレスへ戻るときは、ハイフン(-)をタイプします。

4. EBコマンドを終了するときは、リターンキーを押します。

サンプル

```
-EB CS：100 2D E5 3
```

この例では、CS：100 に 2DH、CS：101 に E5H、CS：102 に 03H を入力したことになります。

```
-EB ES：100
```

このように入力すると、つぎのような形式でプロンプトを表示します。

```
04BA：0100   EB.
```

ここで、41 とタイプすると、値 EB を新しい値 41(H)に変更できます。

```
04BA：0100   EB.41
```

スペース キーを押すと、次のバイトへ進みます。

```
04BA：0100   EB.41   10.
```

ハイフンをタイプすると、前の値へ戻ることができます。

```
04BA：0100   EB.41   10.-
04BA：0100   41.
```

SYMDEB

# EW

## Enter Words

## 指定したアドレスへワード値を入れる

**書　式**　EW <アドレス>[<数値>]

### 解　説

指定したアドレスにワード値(2バイト)を書き込みます。<数値>には、ワード値をスペースまたはカンマで区切って複数指定できます。

<数値>をつけずに、アドレスだけを指定すると、そのアドレスから始まる1ワード分の現在のメモリの値を表示し、ユーザーの入力を待ちます。ここで、4桁以内の16進数で値を入力し、リターンキーを押します。単にリターンキーを押すと、EWコマンドを終了します。EAやEBコマンドのように、アドレスを進めたり戻ったりすることはできません。

**サンプル**

```
-EW  CS:400
2344:0400    ED32.8AD8
2344:0402    1D3C.
```

この例では、〝2344：0400〟とその内容〝ED32〟が表示されているので、〝8AD8〟をタイプしています。さらに、次のアドレスとその内容を表示していますが、この状態でリターンキーを押すと、EWコマンドは終了します。

# ED

## Enter Doublewords

## 指定したアドレスへダブルワード値を入れる

**書　式**　　ED <アドレス>[<数値>]

### 解　説

指定したアドレスにダブルワード値(4 バイト)を書き込みます。<数値>には、ダブルワード値をスペースまたはカンマで区切って複数指定できます。

<数値>をつけずに、アドレスだけを指定すると、そのアドレスから始まる1ダブルワード分の現在のメモリの値を表示し、ユーザーの入力を待ちます。ここで、8桁以内の16進数で値を入力し、リターンキーを押します。単にリターンキーを押すと、EDコマンドを終了します。EAやEBコマンドのように、アドレスを進めたり戻ったりすることはできません。

**サンプル**

```
-ED CS:100
2344:0100     440E:1234.1234:5678
2344:0104     8ED9:1234.
```

この例では、アドレス"2344:0100"とその内容"440E:1234"が表示されているので、"1234:5678"をタイプしています。さらに、次のアドレスとその内容を表示していますが、この状態でリターンキーを押すと、EDコマンドは終了します。

# ES　Enter Short Reals

## 指定したアドレスにショート実数(4バイト)を入れる

**書式**　ES <アドレス>[<数値>]

### 解説

指定したアドレスへ4バイト形式の浮動小数点値(実数)を書き込みます。＜数値＞には、ショート実数をスペースまたはカンマで区切って複数指定できます。

＜数値＞をつけずにアドレスだけを指定すると、そのアドレスから始まる4バイト分の現在のメモリの値を浮動小数点表示で示し、ユーザーの入力を待ちます。

＜数値＞を指定すると、指定されたアドレス(4バイト分)の内容を変更し、続いて次のアドレスとその現在の値を表示し、ユーザーの入力を待ちます。ESコマンドを終了するときは、リターンキーを押します。

**サンプル**

```
-ES pi 3.1415926
```

この例は、シンボリックアドレス"pi"へ、3.1415926という数値をショート実数の形式で入れたものです。ESコマンドで、アドレスだけを指定した場合は、つぎのようになります。

```
-ES pi
210C:0130    -0.1256210825216E+16     +0.31415926e+1
210C:0134    -0.4309309980615894E-31
```

ESコマンドで入れた値を調べるには、DSコマンドを用います。先頭バイトのアドレス、4バイトの内容、浮動小数点形式で表示した10進数の数値が表示されます。この数値は、16桁で表示されますが、有効数字は小数点以下7桁までで、後ろの9桁は意味を持ちません。

# EL

## Enter Long Reals

### 指定したアドレスにロング実数(8 バイト)を入れる

**書　式**　EL <アドレス>[<数値>]

## 解　説

指定したアドレスへ8バイト形式の浮動小数点値(実数)を書き込みます。<数値>には、ロング実数をスペースまたはカンマで区切って複数指定できます。

<数値>をつけずにアドレスだけを指定すると、そのアドレスから始まる8バイト分の現在のメモリの値を浮動小数点表示で示し、ユーザーの入力を待ちます。

<数値>を指定すると、指定されたアドレス(8バイト分)の内容を変更し、続いて次のアドレスとその現在の値を表示し、ユーザーの入力を待ちます。ELコマンドを終了するときは、リターンキーを押します。

**サンプル**

```
-EL pi 3.141592653589793
```

この例は、シンボリックアドレス"pi"に、数値"3.141592653589793"をロング実数の形式で入れたものです。ELコマンドで、アドレスだけを指定した場合は、つぎのようになります。

```
-EL 210C:0170
210C:0170  +0.1343280735843091E+65299  +0.3141592653589793e+1
210C:0178  +0.1040230032441619E-71
```

# ET

## Enter Ten-byte Reals

## 指定したアドレスに10バイト実数を入れる

**書　式**　ET <アドレス>[<数値>]

### 解　説

指定したアドレスへ10バイト形式の浮動小数点値(実数)を書き込みます。<数値>には、実数をスペースまたはカンマで区切って複数指定できます。

<数値>をつけずにアドレスだけを指定すると、そのアドレスから始まる10バイト分の現在のメモリの値を浮動小数点表示で示し、ユーザーの入力を待ちます。

<数値>を指定すると、指定されたアドレス(10バイト分)の内容を変更し、続いて次のアドレスとその現在の値を表示し、ユーザーの入力を待ちます。ETコマンドを終了するときは、リターンキーを押します。

**サンプル**

```
-ET pi 3.141592653589793
```

この例は、シンボリックアドレス"pi"へ、数値"3.141592653589793"を入れたものです。

```
-ET pi
210C:0150　+0.0204654128113587E+7898　+0.314152653589793e+1
210C:015A　+0.5976239733286124E+3896
```

この例は、ETコマンドで、アドレスだけを指定した場合です。

# E　Enter

## メモリの内容を変更する

**書　式**　E <アドレス> [<リスト>]

### 解　説

指定した<アドレス>のメモリの内容を変更します。直前に使ったEnter コマンド(EA、EB、EW、ED、ES、EL、ET などのメモリに値を書き込むコマンド)と同じ形式で内容を書き込みます。

<リスト>を指定すると、その値でメモリの内容を書き換えます。<リスト>を指定しないと、アドレスと現在のメモリの値とピリオド(.)を表示して、新しい値の入力待ちになります。

**サンプル**

```
-EW ES：100 ABCD
-E ES：100
04BA：0100   ABCD.1234
04BA：0100   0D0A.
```

SYMDEB

# X　Examine Symbol Map

## シンボルマップ内のシンボル名とアドレスを表示する

**書　式**

X [*]
または
X? [<マップ名>！]
または
X? [<セグメント名>：][<シンボル名>]

## 解　説

現在のシンボルマップとそのマップ内のセグメントの名前およびロードセグメントアドレスを表示します。アスタリスク(*)を指定すると、コマンドはすべてのシンボルマップの名前とロードセグメントアドレスを表示します。

X？コマンドは、シンボルマップ内の１つまたは複数のシンボルの名前とアドレスを表示します。〝<マップ名>！〞を入力すると、コマンドはそのシンボルマップの情報を表示します。マップ名は、シンボルファイルのファイル名(拡張子の〝.SYM〞は不要)です。〝<セグメント名>：〞を入力すると、セグメントの名前とロードセグメントアドレスを表示します。セグメント名は、シンボルマップ内か、現在開かれているシンボルマップ内にあるセグメントの名前でなければなりません。〝<シンボル名>〞を入力すると、コマンドは、そのシンボルのセグメントアドレスとセグメントのオフセットを表示します。シンボル名は、入力されたセグメント内のシンボルの名前でなければなりません。

複数のセグメントやシンボルの情報を表示する場合、セグメント名やシンボル名にアスタリスク(*)をつけることができます。アスタリスクは、ワイルドカード的な文字として働くので、たとえば、〝F＊：〞は、文字〝F〞で始まるすべてのセグメント名であり、〝_*〞は、アンダースコア(＿)で始まるすべてのシンボルを表します。

**サンプル**

```
-X
```

この例では、現シンボルマップの名前と、そのマップ内のセグメントの名前と、ロードセグメントアドレスが表示されます。

```
-X? test!
```

この例では、シンボルマップファイル〝test〟のロードセグメントアドレスが表示されます。

```
-X? igroup :
```

この例では、現在オープンしているシンボルマップ内の〝igroup〟という名前のセグメントのロードセグメントアドレスが表示されます。

```
-X? start
```

この例では、現在オープンしているシンボルマップ内のシンボル〝start〟のセグメントのアドレスと、セグメントのオフセットが表示されます。

```
-X? igroup : start
```

この例では、現在オープンしているシンボルマップ内のセグメント〝igrorp〟のシンボル〝start〟のセグメントのアドレスと、セグメントのオフセットが表示されます。

```
-X? code *
```

この例では、現シンボルマップ内の文字〝code〟で始まるすべてのシンボルのセグメントのアドレス値とオフセット値が表示されます。

SYMDEB

# F

## Fill

### 指定した範囲を指定した値で埋める

**書　式**　F <レンジ> <リスト>

**解　説**

指定した＜レンジ＞内のアドレスのメモリを、＜リスト＞の値で満たします。＜リスト＞の長さが＜レンジ＞よりも短いときは、レンジ内のすべてのバイトが満たされるまで、＜リスト＞が繰り返して用いられます。＜リスト＞の値が＜レンジ＞のバイト数より大きい場合、余分な値は無視されます。

**サンプル**

```
-F CS：100 L 100 FF
```

この例では、メモリのCS：100からCS：1FFまでがバイト値FFHで満たされます。

```
-F DGROUP：table L 64 42 45 52 54 41
```

この例では、〝DGROUP：table〞から100(64H)バイトが、入力されたバイト値で満たされます。＜リスト＞の5個の値は、100バイトすべてが一杯になるまで繰り返されます。

# G

## Go

## プログラムを実行する

**書　式**　G [=<開始アドレス>] [[<ブレークポイントアドレス>]…]

### 解　説

指定した<開始アドレス>からプログラムを実行します。プログラムは、終わるまで、または<ブレークポイントアドレス>に達するまで実行されます。また、プログラムはBPコマンドでセットされたブレークポイントでも停止します。

<開始アドレス>を省略すると、コマンドは、現在のCS：IP値で示されるアドレスから実行を開始します。この場合の実行開始アドレスは、つぎのようになります。

- 直前にGコマンドを実行している場合は、直前のGコマンドの終了アドレスの次のアドレスから開始されます。
- SYMDEB起動時、またはLコマンドでファイルをロードした場合は、CS：0100H番地から開始されます。ただし、ロードしたファイルがEXE形式の場合は、CS：IPの値がプログラム先頭にセットされているため、そのアドレスから開始されます。
- RコマンドでCS：IPの値を変更した場合は、そのアドレスから開始されます。

なお、<開始アドレス>を指定するときは、必ずイコール記号(=)をつけます。

<ブレークポイントアドレス>には、命令コードの最初のバイトを指定します。最大10個のブレークポイントアドレスを設定できます。ブレークポイントに達すると、すべてのレジスタとフラグの現在値を表示します。また、次に実行する命令の表示も行います。

なお、Gコマンドの表示形式は、Set Source Modeコマンドの設定により、逆アセンブルモード、ソースモード、ミックスモードの3種類を選択できます。

**注意**　Gコマンドは、IRET命令を用いてプログラムに制御を渡すために、ユーザースタックは最低6バイト必要です。ユーザースタックがこれより少なかったり、無効なメモリ内にある場合、MS-DOSのシステムメモリを破壊することもあります。

ブレークポイントを設定すると、そのアドレスにINT 03H命令（割り込みコード0CCH）を置き、ブレークポイントに達すると、これを元の命令に復元します。しかし、実行がブレークポイントを通らずにプログラムの最後まで続けられたり、何か他の理由で中止された場合、割り込みコードの置換は行いません。このため、プログラムを再開させる前にNコマンドとLコマンドを用いて、プログラムを再ロードしなくてはなりません。

プログラムが最後まで実行された場合、〝Program terminated normally〞というメッセージが表示されます。

SYMDEB

サンプル

```
-G = CS : 0 CS : 7550
```

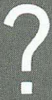
この例では、アドレス"CS：0"のプログラムに実行の制御が渡されます。SYMDEBは、ブレークポイントアドレス"CS：7550"に達すると、実行を中止してレジスタとフラグの現在の値を表示します。

```
-G
```

この例では、CSレジスタとIPレジスタの現在の値が示す命令へ制御が渡されます。このコマンドは一般に、ブレークポイントに達した後の実行の継続に用いられます。

```
-G =_main add
```

この例では、シンボルアドレス"_main"が指すアドレスに制御が渡されます。ブレークポイントは、アドレス"add"にセットされます。



# Help

## SYMDEB コマンドの一覧を表示する

**書　式**　?

## 解　説

SYMDEB のコマンドの一覧を表示します。表示されるリストをつぎに示します。

A[〈address〉] - assemble
BC[〈bp〉] - clear breakpoint(s)
BD[〈bp〉] - disable breakpoint(s)
BE[〈bp〉] - enable breakpoint(s)
BL[〈bp〉] - list breakpoint(s)
BP[bp] 〈address〉 - set breakpoint
C〈range〉 〈address〉 - compare
D[type][〈range〉] - dump memory
E[type] 〈address〉 [〈list〉] - enter
F〈range〉 〈list〉 - fill
G[=〈address〉 [〈address〉...]] - go
H〈value〉 〈value〉 - hexadd
I〈value〉 - input from port
K[〈value〉] - stack trace
L[〈addr〉 [〈drive〉〈rec〉〈rec〉]] - load

M〈range〉 〈address〉 - move
N〈filename〉 [〈filename〉...] - name
O〈value〉 〈byte〉 - output to port
P[=〈address〉] [〈value〉] - program step
Q- quit
R[〈reg〉] [[=] 〈value〉] - register
S〈range〉 〈list〉 - search
S{-| &| +} - source level debugging
T[=〈address〉] [〈value〉] - trace
U[〈range〉] - unassemble
V[〈range〉] - view source lines
W[〈address〉 [〈drive〉〈rec〉〈rec〉]] - write
X[?] 〈symbol〉 - examine symbols(s)
XO〈symbol〉 - open map/segment
Z〈symbol〉 〈value〉

? 〈expr〉 - display expression
! [dos command] - shell escape
.- display current source line
¥- screen flip

〉 } 〈device/file〉 - Redirect output
〈 { 〈device/file〉 - Redirect input
= ~ 〈device/file〉 - Redirect both
* 〈string〉 - comment

〈expr〉 ops　:　+ - * /　:　not seg off by wo dw poi port wport mod and xor or
〈type〉　:　Byte, Word, Doubleword, Asciz, Shortreal, Longreal, Tenbytereal

SYMDEB

# H

## Hex

### 2つの16進数の和と差を計算する

**書　　式**　　H <数値 1> <数値 2>

## 解　　説

＜数値 1＞と＜数値 2＞を加算して結果を表示し、次に＜数値 1＞から＜数値 2＞を引いてその結果を表示します。もし＜数値 2＞が＜数値 1＞より大きければ、2 つの値の差を 2 の補数として表示します。

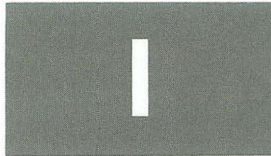
**サンプル**

```
-H 3 4
```

この例では、7 と FFFF という結果が表示されます。

```
-H 110 100
```

この例では、210 と 10 という結果が表示されます。




# Input

## 指定したポートから1バイトを読み込んで表示する

**書　　式**　　I <ポート>

### 解　　説

指定したI/Oポートから1バイトを読み込んで、2桁の16進数で表示します。

**サンプル**

```
-I 2F8
```

この例では、I/Oポート番号"2F8"から読み取ったバイト値が表示されます。



# Load

## ファイルまたは指定したレコードを読み込む

**書　　式**

L [<アドレス>]
または
L <アドレス> <ドライブ> <レコード> <カウント>

### 解　　説

N コマンドで指定されたファイルや、ディスク中の指定レコードのデータを、メモリに読み込みます。

**書式 1：　L [<アドレス>]**

N コマンドまたは SYMDEB 起動時のコマンドラインで指定したファイルを、指定したアドレスにロードします。＜アドレス＞を省略すると CS：100 からロードされます。ロードされたファイルの長さは、BX：CX レジスタにセットされます。

指定したファイルが COM 形式（拡張子が".COM"）の場合、必ず CS：100 からロードされるので、＜アドレス＞を指定しても無効です。

指定したファイルが EXE 形式（拡張子が".EXE"）の場合、EXE ファイルのヘッダで指定されるアドレスにロードされ CS：IP の値はそのロードアドレスにセットされます。したがって、EXE ファイルの場合には、＜アドレス＞の指定は無効です。

指名したファイルが HEX 形式（拡張子が".HEX"）の場合、そのファイルの開始アドレスと＜アドレス＞を加算したアドレスにロードします。＜アドレス＞を指定しないと、ファイルはその開始アドレスにロードされます。

**書式 2：　L <アドレス> <ドライブ> <レコード> <カウント>**

ディスクから直接、論理レコードをメモリにロードします。このときは、＜アドレス＞、＜ドライブ＞、＜レコード＞、＜カウント＞の値を明確に指定します。ドライブは読み込む先のドライブ番号で、ドライブ A：なら 0、ドライブ B：なら 1、ドライブ C：なら 2…というように指定します。＜レコード＞は、ディスクから読み取る最初の論理レコード番号で、1～4 桁の 16 進数です。＜カウント＞は、ディスクから読み取るレコード数を、1～4 桁の 16 進数で指定します。

サンプル

```
-N file.com
-L
```

この例では、"file.com"という名前のファイルが、メモリのアドレス CS：100 にロードされます。ロードされた長さ(バイト数)は、BX：CX レジスタペアに収められます。

```
-L DGROUP：table
```

この例では、シンボルアドレス"DGROUP：table"からファイルがロードされます。N コマンドかコマンドラインで指定されたファイルをロードします。

```
-L workspace 2 34 3
```

この例では、ドライブ C：(2)の論理レコード番号 52(34H)から始まる 3 つの論理レコードがメモリのシンボルアドレス"workspace"へロードされます。

# M

## Move

### 指定したメモリのブロックを移動(コピー)する

**書　式**　M <レンジ> <アドレス>

### 解　説

＜レンジ＞で指定したメモリのブロックを、＜アドレス＞から始まる位置へ移動します。転送元と転送先が重なっていてもデータを損失することなく移動します。＜レンジ＞で指定したアドレスのメモリは、Mコマンド実行後もそのまま残っています。ただし、転送先が転送元と重なった場合は、転送元が変更されます。

**サンプル**

```
-M CS:100 110 CS:500
```

この例では、CS：100からCS：110までのブロックを、CS：500から始まるメモリに移動します。

```
-M DS:table L 100 workspace
```

この例では、シンボルアドレス〝DS：table〟の256(100H)バイトをアドレス〝workspace〟に移動します。

# N

## Name

### ファイル名などの引数をセットする

**書　式**　N <引数> [[<引数>]…]

**解　説**

Lコマンドや W コマンドのためにファイル名をセットしたり、プログラムの実行のために引数をセットしたりします。

<引数>を入力すると、スペースを含むすべての引数を DS：81 から始まる位置にコピーして、DS：80 にコピーした文字数をセットします。

最初の 2 つの引数がファイル名である場合、アドレス DS：5C とアドレス DS：6C にそれぞれファイルコントロールブロック(FCB)を作成して、このブロックに名前(正しい形式の名前)をコピーします。これによって、デバック中のプログラムが FCB を使用することができます。

N コマンドによって影響を受けるメモリのアドレスはつぎのとおりです。

| | |
|---|---|
| DS：5C | 1番目のファイル用 FCB |
| DS：6C | 2番目のファイル用 FCB |
| DS：80 | 文字の総数 |
| DS：81 | 入力されたすべての文字 |

N コマンドによっていったん引数をコピーすれば、デバッグ中のプログラムによってアクセスが可能となります。

**注意**　Load や Write 用に新しいファイル名をセットすると、前のプログラム引数は破壊されてしまいますので注意が必要です。

SYMDEB

**サンプル**

```
-N file.exe
```

この例では、以後のLコマンドやWコマンドのために、ファイル名を〝file1.exe〃にセットします。

```
-N file.com test.dat
-L
```

この例では、〝test.dat〃引数をセットして、〝file.com〃を読み込みます。ここで、

```
-G
```

として実行すると、これは、MS-DOSのコマンドラインから、

```
A>file test.dat
```

と実行した結果と同じになります。

# XO

## Open Map

### シンボルマップまたはセグメントをセットする

**書　式**　XO [<マップ名>!]<セグメント名>

## 解　説

〝＜マップ名＞！″で指定したシンボルマップをアクティブにします。＜マップ名＞は、SYMDEB コマンドラインに指定したシンボルファイルの中の１つでなくてはなりません。

＜セグメント＞名を指定すると、指定されたセグメントをアクティブにします。＜セグメント名＞は、＜マップ名＞で指定したシンボルマップファイル内、または現在のシンボルマップ内の、セグメントの名前でなければなりません。

**サンプル**

```
-XO test!
```

この例では、シンボルマップ〝test″がアクティブになります。

```
-XO test!group
```

この例では、シンボルマップ〝test″とそのセグメント〝igroup″がアクティブになります。

```
-XO dgroup
```

この例では、現在のシンボルマップのセグメント〝dgroup″がアクティブになります。

# O

## Output

### I/O ポートへ出力する

**書　式**　O <ポート> <バイト>

## 解　説

指定した＜バイト＞を指定した＜ポート＞に出力します。＜ポート＞は 16 ビットのポートアドレスです。

**サンプル**

```
-O 2F8 4F
```

この例では、バイト値 4F(16 進)が出力ポート 2F8 へ送られます。

```
-O 3 21
```

この例では、バイト値 21(16 進)が出力ポート 3 に送られます。

# P　P Trace

## 割り込みに対応してトレースする

**書　式**　　P [=<開始アドレス>] [<カウント>]

### 解　説

指定した＜開始アドレス＞から、＜カウント＞で指定されたステップ数だけ、実行中のレジスタとフラグを表示します。

＜開始アドレス＞を省略すると、現在の CS レジスタと IP レジスタが示す命令から実行を開始します。イコール記号(=)は、開始アドレスを入力する場合にのみ使います。

＜カウント＞を省略すると、1 ステップだけトレースします。

なお、P コマンドの表示形式は、Set Source Mode コマンドの設定により、逆アセンブルモード、ソースモード、ミックスモードの 3 種類を選択できます。

**注意**　P コマンドは T コマンドとよく似ていますが、たとえば CALL 命令があった場合、P コマンドはそのサブルーチンを実行して、サブルーチン内の命令をステップ数としてカウントしません。一方、T コマンドはサブルーチン内の命令もステップ数として数えます。つまり、サブルーチンコール、ソフトウェア割り込み、ループなどのあるプログラムを P コマンドと T コマンドでトレースした場合、停止する位置が異なることになります。

**サンプル**

```
-P  =_main
```

この例では、〝_main〟の命令が実行され、レジスタとフラグの現在値が表示されます。また、次に実行する命令も表示されます。

```
-P
```

この例では、現 CS レジスタと IP レジスタが示す命令が実行されます。

# Q

## Quit

SYMDEBを終了する

**書　　式**　Q

### 解　　説

SYMDEBを終了し、MS-DOSに戻ります。このとき、デバッグ中のプログラムはディスクに書き込まれません、必要であればNコマンドでファイル名を指定してから、Wコマンドで書き込んでください。




## Redirection

### 入出力の切り換えを行う

**書　式**

〈 <装置名>
〉 <装置名>
= <装置名>

## 解　説

SYMDEB の入力と出力を＜装置名＞で指名した装置に切り換えます。＜装置名＞には、デバイス名またはファイル名を指定します。

〈 コマンドは、指定した＜装置名＞からコマンド入力を読み込みます。

〉 コマンドは、指定した＜装置名＞にコマンド出力を書き出します。

= コマンドは、指定した＜装置名＞に対して読み込みと書き出しの両方を行います。

装置名には、つぎのようなものがあります。詳しくは「MS-DOS ユーザーズリファレンスマニュアル」を参照してください。

| | |
|---|---|
| AUX | 補助入出力装置 |
| CON | コンソール |

補助入出力装置(AUX)を用いる場合、ボーレートなどの設定や初期化を行っておく必要があります。

一般に、Redirection コマンドは、スクリーンを全面的に使用するプログラムをデバッグする場合に使います。

**注意**　入力をコンソール以外に切り換えると、CTRL ＋ S キーと CTRL ＋ C キーは無効となります。

SYMDEB

**サンプル**

```
-> AUX
```

この例では、SYMDEBコマンドの出力が“AUX”の装置へ切り換えられます。

```
-= AUX
```

この例では、コマンドの入力と出力が“AUX”の装置へ切り換えられます。

```
-< file.dat
```

この例では、あらかじめ“file.dat”に書いておいたSYMDEBのコマンドを入力させ、それを実行させます。

# R

## Register

## レジスタ値を表示、設定する

**書　式**　R [<レジスタ名> [<数値>]]

### 解　説

レジスタに格納されている値を表示したり、逆にレジスタに値を入力したりします。

＜レジスタ名＞を省略すると、現在のレジスタおよびフラグと、CS レジスタと IP レジスタが示す命令を表示します。

＜レジスタ名＞を指定すると、指定したレジスタの現在の値を表示し、新しい値の入力を求めてきます。

＜レジスタ名＞と＜数値＞の両方を指定すると、レジスタを指定された値に変更します。

＜レジスタ名＞は、つぎに示すうちの1つです。

| | | |
|---|---|---|
| AX | BP | SS |
| BX | SI | CS |
| CX | DI | IP |
| DX | DS | PC |
| SP | ES | F |

IP と PC は同じレジスタ、つまり命令ポインタを指名するものです。F は、フラグレジスタの名前です。

レジスタの値を変更する場合は、レジスタ名と値を入力します。レジスタだけ指定して値を入力しないと、レジスタ名、現在の値、そしてコロン(:)のプロンプトが表示されます。新しい値をタイプしてリターンキーを押してください。値を変更したくない場合は、リターンキーだけを押します。不適切なレジスタ名を入力すると、〝Bad Register〟というメッセージが表示されますので、入力し直してください。

フラグの値を変更する場合は、レジスタ名として〝F〟を入力します。コマンドは、各フラグの現在値を2文字略号で表示します。フラグ値の略号のリストをつぎに示します。

| フラグ | セット | クリア |
|---|---|---|
| オーバーフロー | OV | NV |
| 方向 | DN(減少) | UP(増加) |
| 割り込み | EI(有効) | DI(無効) |
| 符号 | NG(負) | PL(正) |
| ゼロ | ZR | NA |
| 補助桁上げ | AC | NA |
| パリティ | PE(偶数) | PO(奇数) |
| 桁上げ(キャリー) | CY | NC |

〝F″を指定すると、フラグ値とハイフン(-)が表示されます。ハイフンが示されたら、変更したいフラグの新しい値を入力してリターンキーを押します。フラグの値の入力順序は任意です。値の間にスペースは必要ありません。新しい値の入力されないフラグは、変更されません。どのフラグも変更したくない場合は、リターンキーだけを押します。

＜レジスタ名＞として〝F″を指定した場合は、その後に＜数値＞を指定しても無視されます。

1つのフラグに対して複数の値を入力すると、〝Double flag!″というメッセージが表示されます。上記以外の名前を入力すると、〝Bad Flag!″というメッセージが表示されます。どちらの場合も、エラーが発生するまでのフラグの変更は行われますが、エラーのフラグとそれ以後のフラグは変更されません。

なお、Rコマンドの表示形式は、Set Source Modeコマンドの設定により、逆アセンブルモード、ソースモード、ミックスモードの3種類を選択できます。

**サンプル**

```
-R
AX=0E00  BX=00FF  CX=0007  DX=01FF  SP=039D  BP=0000
SI=005C  DI=0000  DS=04BA  ES=04BA  SS=04BA  CS=04BA
IP=011A  NV  UP  DI  NG  AC  PE  NC
04BA:011A  CD21  INT    21
```

この例では、CSレジスタとIPレジスタの指すアドレスの命令に加えて、すべてのレジスタとフラグの値を表示します。命令は最後に表示されます。

```
-R AX
```

この例では、AXレジスタの現在値と、新しい値に対するプロンプトをつぎのように表示します。

```
AX 0E00
:
```

コロン(：)の後に 16 ビットの値をタイプします。変更したくない場合には、リターンキーを押します。

```
-RF
```

この例では、現在のフラグ値と変更を求めるプロンプトをつぎのように表示します。

```
NV  UP  DI  NG  NZ  AC  PE  NC  –
```

フラグ値の変更には、このプロンプトの方法を用いてください。

```
-R IP 100
```

この例では、IP レジスタが値 100 に変更されます。

# S

## Search

### バイト値を検索(サーチ)する

**書　　式**　　S <レンジ> <リスト>

### 解　　説

指定した＜レンジ＞のメモリから＜リスト＞で指定したバイト値を検索します。見つかると、そのアドレスを表示します。そうでない場合は、何も表示せず、SYMDEB のコマンドプロンプトを表示します。

＜リスト＞には1つ以上のバイト値またはストリングを指定します。複数のバイト値を指定する場合は、スペースかカンマで区切らなければなりません。文字列の場合はシングルクォーテーション(')かダブルクォーテーション(")で囲みます。

**サンプル**

```
-S CS：100 200 41
```

この例では、CS：100 から CS：200 の範囲内でバイト値 41 を検索します。

```
-S table L 100 "Fix up"
```

この例では、シンボルアドレス〝table〟から始まる 256(100H)バイトの範囲で〝Fix up〟という文字列を検索します。

# S

## Set Source Mode

## 命令コードの表示形式を設定する

**書　　式**　S{-|+|&}

### 解　　説

Gコマンド、Pコマンド、Rコマンド、TコマンドおよびUコマンドが表示する形式を設定します。以下の3つのモードを設定できます。

書式1：　S-
逆アセンブルモード。命令コードのみが表示されるモードとなります。

書式2：　S+
ソースモード。表示すべき命令に対応する実際のプログラムのソース行が表示されるモードとなります。

書式3：　S&
ミックスモード。命令コードとソース行の両方が表示されるモードとなります。

ただし、ソース行は、S+コマンドかS&コマンドが入力されていて、さらに行番号の情報を持つシンボルファイルがロードされている場合に限って表示することができます。シンボルファイルをロードしていなかったり、シンボルファイルに行番号の情報が入っていないと、以後のソース行表示の要求を無視します。

ソース行は"行番号：ソース"の形式で表示され、命令コードが表示される前に表示されます。

**サンプル**

```
-S+
```

この例では、ソース行のみの表示形式にセットします。

```
-S&
```

この例では、ソース行と命令コードの表示形式にセットします。



# Shell Escape

## SYMDEB中から、MS-DOSコマンドを実行する

**書　式**　!<MS-DOSコマンド>

### 解　説

SYMDEBの中から、<MS-DOSコマンド>で指定したMS-DOSのコマンドを子プロセスとして実行します。<MS-DOSコマンド>を省略すると〝COMMAND.COM〟のみを実行するため、MS-DOSのプロンプトが表示されます。この状態では、通常のMS-DOSモードと同様に操作を行うことができます。SYMDEBに戻るときは、〝EXIT〟コマンドを用います。

**参考**

！(Shell Escape)コマンドを実行するためには、デバッグ中のプログラムが不必要なメモリを開放するようになっていなくてはいけません。不必要なメモリを開放するためには、プログラムでMS-DOSのファンクションコール4AH(割り当てられたメモリブロックの変更)を用います。このファンクションコールによって、MS-DOSは、COMMAND.COMをロードするメモリ領域を確保することができます。同様のことが、LINK(リンカ)の/CPARMAXALLOCスイッチでも行えます。

Microsoft C Ver3.0以上で開発したプログラムで、_main関数の実行後であれば、自動的にメモリは開放されます。Microsoft Pascal、FORTRAN Ver3.30以上で開発されたプログラムでも、最初の手続きの実行後であれば、メモリは自動的に開放されます。SYMDEB自身をロードした場合でも、メモリは開放されます。しかし、MASMまたは前述のコンパイラと同様の機能を持たないコンパイラで開発したプログラムのデバッグを行うときは、メモリの開放を行わないと！(Shell Escaspe)コマンドは実行できません。

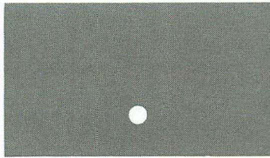
**サンプル**

```
-! dir b：*.asm
```

この例は、MS-DOSのコマンドモードで〝dir b：*.asm〟を実行したときと同じように表示され、その後再びSYMDEBのコマンド入力モードに戻ります。

```
-! chkdsk b：
```

この例は、MS-DOSの外部コマンド〝CHKDSK〟を実行するものです。このように外部コマンドやバッチファイルも実行できます。



# Source Line

## 現在のソース行を表示する

**書　式**　.（ピリオド）

### 解　説

Source Lineコマンド(.)は、高級言語によるプログラムのデバッグに用いるコマンドです。たとえば、プログラムをブレークポイントで停止させた時点で、ピリオド(.)を入力すると、そのときのソースモード(Set Source Modeコマンド参照)に関係なく、現在のソース行を表示します。

なお、このコマンドは、MASM、SYMDEBと互換性の無いコンパイラで開発したプログラムでは無効です。

**注意**　ソース行に2バイト文字が含まれている場合、表示が乱れることがあります。

**サンプル**

```
-.
for  (i = 0 ; i<= SIZE ; i++) ;
```

この例は、SYMDEBと互換性のあるCコンパイラで開発したプログラムをデバッグ中のものです。

SYMDEB

# K　Stack Trace

## スタックフレームの内容を表示する

**書　　式**　　K [<値>]

## 解　　説

Kコマンドは、現在のスタックフレームの内容を表示します。表示される項目は、現在の手続きと引数、コールされている手続きおよび引数です。さらに、各手続きをコールしているファイルとソース行の行番号も表示されます。

パラメータ＜値＞が指定されると、その値の数まで引数が16進数で表示されます。

**サンプル**

```
-K
IGROUP：_max(0002,0003)from .sample.c：7
IGROUP：_main(?)
```

この例では、現在の手続きは〝_max〟で、2つの引数〝0002H〟と〝0003H〟であることが分かります。また、この手続き〝_max〟は、ソースファイル〝sample.c〟の7行目の手続き〝_main〟からコールされており、〝_main〟の引数は不定であることも分かります。

```
-K 3
IGROUP：max(0002,0003)from..sample.c：7
IGROUP：main(0002,2F62,2EBA)
```

この例では、手続き〝_main〟の引数が、3個まで表示されています。

# Z

## Symbol Set

### シンボリックアドレスに値をセットする

**書　式**　　Z <シンボル> <値>

**解　　説**

指定した＜シンボル＞のアドレスへ指定した＜値＞を入れます。

**サンプル**

```
-Z close 26
```

この例では、シンボル〝close〟のアドレスへ 26H をセットしています。

SYMDEB

# T

## Trace

### プログラムの実行をトレースする

**書　　式**　T [=<開始アドレス>] [<カウント>]

## 解　　説

指定した<開始アドレス>から、<カウント>で指定されたステップ数だけ、実行中のレジスタとフラグを表示します。

<開始アドレス>を省略すると、コマンドは、現在のCS：IP値で示されるアドレスから実行を開始します。この場合の実行開始アドレスは、つぎのようになります。

- 直前にTコマンドを実行している場合は、直前のTコマンドの終了アドレスの次のアドレスから開始されます。
- SYMDEB起動時、またはLコマンドでファイルをロードした場合は、CS：0100H番地から開始されます。ただし、ロードしたファイルがEXE形式の場合は、CS：IPの値がプログラム先頭にセットされているため、そのアドレスから開始されます。
- RコマンドでCS：IPの値を変更した場合は、そのアドレスから開始されます。

なお、<開始アドレス>を指定するときは、必ずイコール記号(=)をつけます。

<カウント>を省略すると、1ステップだけトレースします。

なお、Tコマンドの表示形式は、Set Source Modeコマンドの設定により、逆アセンブルモード、ソースモード、ミックスモードの3種類を選択できます。

**参考**

> Tコマンドは、8086/8088/80186/80286/386/486マイクロプロセッサのハードウェアトレースモードを使用するので、ROM(読み取り専用メモリ)に記憶されている命令をトレースすることもできます。

**サンプル**

```
-T =_main
```

この例では、_mainからの命令が実行され、レジスタとフラグの現在値を表示します。また、次に実行する命令も表示します。

```
-T
```

この例では、現在のCSレジスタとIPレジスタの示す命令が実行されます。

```
-T = 011A 10
```

この例では、現在のCSセグメントの011Aから、16(10H)ステップだけ命令が実行されます。

# U　Unassemble

## メモリ内容を逆アセンブルする

**書　式**　U [<レンジ>]

### 解　説

デバッグ中のプログラムの命令、プログラム文、またはその両方を、指定された＜レンジ＞の範囲分だけ表示します。＜レンジ＞を省略すると、8行分だけ表示します。表示形式は、Set Source Mode コマンドの設定により、つぎの3種類を選択できます。

**形式1：　逆アセンブルモード（S-）**

命令コードの表示。指定された＜レンジ＞の範囲を逆アセンブルして表示します。その形式は、A コマンドと同じです。シンボルマップがプログラムと共にロードされている場合、ラベルや変数や絶対シンボルを表すオペランドは、アドレスではなく名前で表示されます。

**形式2：　ソースモード（S+）**

プログラムのソース行の表示。指定された＜レンジ＞に対応するソース行を表示します。

**形式3：　ミックスモード（S&）**

ソース行と命令コードの表示。逆アセンブルモードとソースモードをミックスしたものです。

ソースモードとミックスモードの場合、シンボルマップがプログラムと共にロードされることと、ソースファイルの行番号情報がそのマップ内にあることが必要です。プログラムの入力された部分に対する行番号情報がない場合、ソース行は表示されません。

逆アセンブルモードとミックスモードを用いる場合は、SYMDEB は、8ビットの即時オペランドの16進値と ASCII 値の両方を表示します。16進値は命令の一部として表示され、ASCII 値は同じ表示行中のセミコロン(；)の直後に表示されます。

8086と8087の命令コードの表示のみが可能です。

**注意**　　ソース行に2バイト文字が含まれている場合、表示が乱れることがあります。

**サンプル**　　形式1：　逆アセンブルモード

```
-U CS：02AD
```

この例では、アドレス〝CS：02AD″から8行分を逆アセンブルしてつぎのように表示します。

```
1156：02AD  55            PUSH   BP
1156：02AE  8BEC          MOV    BP,SP
1156：02B0  B802C0        MOV    AX,0002
1156：02B3  E893FF        CALL   chkstk
1156：02B6  C746FE6100    MOV    Word Ptr[BP-02],0061
1156：02BB  FF0EEC05      DEC    Word Ptr[05EC]
1156：02BF  833EEC0500    CMP    Word Ptr[05EC],+00
1156：02C4  7C11          JL     02D7
```

形式2：　ソースモード

```
-U _main L 5
```

この例では、アドレス〝_main″から5行分のソース行をつぎのように表示します。

```
4：  {
5：      int i；
6：
7：      for(i ='a'；i<'z'；i++)
8：          putchar(i)；
```

形式3：　ミックスモード

```
-U CS：02 AD
```

この例では、〝CS：02AD″から8行分の命令コードとプログラムソースコードをつぎのように表示します。

```
4：  {
IGROUP：_main
1156：02AD  55            PUSH   BP
1156：02AE  8BEC          MOV    BP,SP
1156：02B0  B80200        MOV    AX,0002
1156：02B3  E893FF        CALL   chkstk
7：      for(i ='a'；i<'z'；i++)
1156：02B6  C746FE6100    MOV    Word Ptr[BP-02],0061
```

SYMDEB

# V　View

## 指定したアドレスからソース行を表示する

**書　式**　　V [<アドレス>]

### 解　説

V コマンドは、高級言語によるプログラムに用いるコマンドで、ソースモードの設定(Set Source Mode コマンド参照)に関係なく、指定したアドレスから始まるソース行を表示します。

このコマンドを用いるためには、ソースプログラムファイル中の行番号に関する情報を持ったシンボルファイルが必要です。MASM、SYMDEB と互換性の無いコンパイラで開発したプログラムではこのコマンドは無効です。

**注意**　ソース行に 2 バイト文字が含まれている場合、表示が乱れることがあります。

**サンプル**

```
-V _func
4: {
5:     int i;
6:
7:     for(i='a'; i<'z'; i++)
8:         putchar(i);
9:     for(i='A'; i<'Z'; i++)
10:        putchar(i);
11:    for(i='0'; i<'9'; i++)
```

この例は、C コンパイラによるもので、アドレス〝_func〟から始まるソース行を表示したものです。同様のことを、FORTRAN や Pascal で開発したプログラムでも行うことができます。

# W

## Write

### ファイルまたはメモリ内容をディスクに書き込む

**書　式**

W [<アドレス>]
または
W <アドレス> <ドライブ> <レコード> <カウント>



## 解　説

N コマンドで指定されたファイルや、ディスク中の指定レコードに、メモリ中のデータを書き込みます。

書式 1：　W [<アドレス>]

指定した＜アドレス＞から、BX：CX レジスタにセットされた長さのメモリ内容をディスクに書き込みます。＜アドレス＞を省略すると CS：100 から書き込みます。

W コマンドを使う前に、R コマンドでファイルの長さを BX：CX レジスタにセットし、N コマンドでファイル名を指定しておく必要があります。ただし、SYMDEB 起動時、または L コマンドでファイルをロードした場合、それらの値は自動的にセットされるため、値をとくに変更しない限り、改めてセットする必要はありません。

なお、EXE 形式および HEX 形式のファイルをロードしている場合、W コマンドで書き込みをすることはできませんので注意してください。

書式 2：　W <アドレス> <ドライブ> <レコード> <カウント>

ディスクの論理レコードに、メモリ内容を直接書き込みます。このときは、＜アドレス＞、＜ドライブ＞、＜レコード＞、＜カウント＞の値を明確に指定します。ドライブは書き込む先のドライブ番号で、ドライブ A：なら 0、ドライブ B：なら 1、ドライブ C：なら 2…というように指定します。＜レコード＞は、ディスクに書き込む最初の論理レコード番号で、1〜4 桁の 16 進数です。＜カウント＞は、ディスクに書き込むレコード数を 1〜4 桁の 16 進数で指定します。書き込まれるメモリ内容のバイト数は、この＜カウント＞で指定されたレコード数に相当する分となります。

**注意**　　これは、ディスクの内容を破壊する可能性のあるコマンドなので、十分注意して使ってください。

**参考**

> Gコマンド、Pコマンド、Tコマンドが実行された場合、ロードされたファイルのファイル名、長さおよび開始アドレスを正しい値にセットしなくてはなりません。また、Rコマンドで BX レジスタと CX レジスタを修正してある場合、これらのレジスタも正しい値にセットしなくてはなりません。

**サンプル**

```
-N table.bin
-R BX
BX 0100
:0000
-R CX
CX D43C
:0100
-W DGROUP:table
```

この例では、アドレス〝DGROUP：table〞からの256(100H)バイトを〝table.bin〞というファイル名で書き込みます。

```
W workspace 2 34 3
```

この例では、アドレス〝workspace〞から始まるメモリの内容を、ドライブC：にセットされたディスクのレコード番号52(34H)から3つ目までの論理レコードに、相当するバイト数分書き込みます。

## 4.6　メッセージ

SYMDEB は、実行できないコマンドや不正なパラメータを見つけると、エラーメッセージを表示します。

通常は、発生したエラーに対して、記号(^)と"Error"というメッセージを表示します。記号(^)は、コマンドライン中のエラーの位置を示すものです。つぎの例は、D コマンドの値の範囲が適切でないことを示しています。

**例：**

```
d cs：100 0
          ^Error
```

また SYMDEB は、エラーメッセージ以外にも、コマンドの実行状況をくわしく伝えるメッセージを表示します。つぎに、それらのメッセージとその意味を示します。



Access denied

リードオンリーファイルに書き込むことはできません。

Bad breakpoint number! (0–9)

ブレークポイントの番号が間違っています。0 から 9 までの数字で指定してください。

Bad Flag!

フラグが間違っています。フラグを変更しようとしましたが、タイプした文字は許容されているフラグ値ではありませんでした。正しいフラグ入力値のリストについては、R コマンドを参照してください。

Bad register

R コマンドで、間違ったレジスタ名が入力されました。正しいレジスタ名のリストについては、R コマンドを参照してください。

Breakpoint list or '*' expected!

BC コマンド、BD コマンド、または BE コマンドで、ブレークポイントのリストを入力せずにコマンドを実行しました。

Double flag!

1 つのフラグに対して 2 つの値を入力しました。指定できるフラグ値は 1 つだけです。R コマンドを参照してください。

Error in EXE or HEX file

ロードした EXE 形式または HEX 形式のファイルが、正しくありません。

Error reading .SYM file

SYMDEB のコマンドラインに指定したシンボルファイルが読み取れませんでした。

EXE and HEX files cannot be written

EXE 形式と HEX 形式のファイルを、ディスクに書き込むことはできません。

EXEC failure

実行したプログラムは、異常終了しました。

File creation error

システムファイル、または隠しファイルに書き込みをしようとしました。ファイルの属性を確認してください。

File not found

指定されたファイルが見つからないため、ロードできません。正しいファイル名を Name コマンドで指定しなおしてから、Load コマンドを実行してください。

Incorrect DOS version

DOS のバージョンが違います。動作可能な DOS のバージョンで起動してください。

Insufficient memory

メモリ容量が足らないため、プログラムを正常に実行できません。

Insufficient space on disk

ディスク容量に十分な空きがないため、書き込むことができません。

Invalid drive specification

指定されたドライブ番号が正しくありません。ドライブ番号は、ドライブ A：なら 0、ドライブ B：なら 1、ドライブ C：なら 2…というように指定します。

Parity error or nonexistant memory error detected

パリティエラーか、または非実装アドレス領域へのアクセスによりメモリエラーが発生しました。

Program terminated and stayed resident

実行したプログラムは、システムメモリ上に常駐終了しました。

Program terminated normally

実行したプログラムは、正常終了しました。

Program terminated with hard error

実行したプログラムは、ハードウェアのエラーにより、異常終了しました。

SYM files cannot be written

SYM ファイルを、ディスクに書き込むことはできません。

Too many breakpoints!

Gコマンドのパラメータとして与えられたブレークポイントの数が10個を越えています。10個以下のブレークポイントでGコマンドを入力し直してください。

Value out of range

指定されたパラメータの値が指定可能な範囲内にありません。正しい値を指定してください。

Writing xx bytes

xxバイト分、ディスクに書き込みました。

## 4.7　SYMDEB の使用できるアセンブラとコンパイラ

SYMDEB/MAPSYM のシンボリックデバッグ機能は、つぎに示す言語で記述されたプログラムに使用できます。

Microsoft FORTRAN version 3.0 以上
Microsoft Pascal version 3.0 以上
Microsoft C version 2.0 以上
Microsoft Macro Assembler version 1.0 以上
Microsoft Basic Compiler version 1.0 以上
Microsoft Business Basic Compiler version 1.0 以上

SYMDEB/MAPSYM のソース行表示機能は、正しい行番号の情報を生成するコンパイラで使用できます。ソース行番号の情報を自動的に生成するコンパイラをつぎに示します。

Microsoft FORTRAN version 3.0 以上
Microsoft Pascal version 3.0 以上

つぎに、ソース行番号の情報を生成する場合にコマンドライン中にスイッチが必要なコンパイラを示します。コンパイラのスイッチについては、言語のマニュアルを参照してください。

Microsoft C version 2.0 以上

# 第5章

# LIB：ライブラリマネージャ

## 5.1 イントロダクション

ライブラリファイルは、他のプログラムが実行時に必要とする"オブジェクトモジュール"の集まりです。ライブラリマネージャ：LIB は、このライブラリファイルの作成、管理を行うためのユーティリティです。

ライブラリは、LINK によって用いられます。LINK は、プログラム中の外部参照を解析し、それをライブラリファイルから捜し出して、参照を解決し、実行可能なプログラムを作成します。

LIB は、"オブジェクトファイル"を取り込んで、ライブラリファイルを作成します。オブジェクトファイルは、1 つのオブジェクトモジュールから成るもので、ユーザーが MASM や高級言語コンパイラで作成します。LIB は、オブジェクトモジュールをライブラリファイルに追加・登録すると、このモジュール名をライブラリファイルの"ライブラリ見出しテーブル"に登録します。LINK は、ライブラリファイルを参照するとき、この見出しテーブルをチェックし、必要なモジュールが登録されているか調べます。目的のモジュールが見つかると、LINK はこれを処理中のプログラムとリンクします。

なお本書では、従来バージョンと同じ機能についてのみ解説しています。

## 5.2 LIB の起動と使い方

LIB を使うときは、ライブラリファイル名、スイッチなどのパラメータを指定します。パラメータの指定方法にはつぎの 3 種類あります。

1. コマンドラインでの指定方法
2. LIB のプロンプトでの指定方法
3. 応答ファイルでの指定方法

パラメータが不足しているときは、必要な情報の入力を求めるコマンドプロンプトを表示します。いずれの方法で LIB を起動した場合でも、CTRL + C キーを押すことでいつでも中止することができます。以下、それぞれの指定方法で LIB を起動する方法を述べていきます。

## ■ コマンドラインでの指定方法

このコマンドの書式は、つぎのとおりです。

LIB　＜ライブラリファイル名＞[/PAGESIZE：n][＜コマンド＞…]
[, リストファイル名, 出力ファイル名]

＜ライブラリファイル名＞

処理の対象とするライブラリファイル名です。新規に作成または編集するライブラリファイル名を指定します。なお、ライブラリファイル名の拡張子が".LIB"の場合は、これを省略することができます。

/PAGESIZE：n

これは、ライブラリのページサイズ(5.2節中の「ライブラリページサイズの設定」参照)を指定するもので、省略した場合は、自動的にページサイズは16となります。

＜コマンド＞

LIBに対する指示です。ライブラリファイルに対して追加、削除などを行うモジュール名やライブラリファイル名に、コマンド文字をつけて指定します。このコマンドについては、5.3「LIBのコマンド」で詳しく解説します。＜ライブラリファイル名＞だけを指定して、コマンドを指定しないと、LIBは、ライブラリの整合性をチェックします。これについては、5.2節中の「ライブラリの整合性のチェック」を参照してください。

＜リストファイル名＞

ライブラリのクロスリファレンスリストファイルを作成したいときにそのファイル名を指定します。このクロスリファレンスリストファイルには、ライブラリ中のモジュールのPUBLICシンボル名が、一覧表になって収められます。この指定を省略するとファイルは作成されません。

＜出力ファイル名＞

処理を行った結果を収める新しいライブラリファイル名を指定します。出力ファイル名を指定しなかった場合、LIBは指定したライブラリファイルそのものを編集しますが、この出力ファイル名を指定すると、編集の対象になったライブラリは元の状態で残され、編集された新しいライブラリファイルが作られます。元のライブラリファイルをそのままの状態で残しておきたいときに指定します。

ライブラリファイルがカレントディレクトリにない場合は、ドライブ名やパス名を含めて＜ライブラリファイル名＞を指定してください。

＜リストファイル名＞を指定するときは、最後の＜コマンド＞の後に、区切り記号のカンマ(,)を入れます。この区切り記号は、＜リストファイル名＞と＜出力ファイル名＞の間にも必要です。＜リストファイル名＞を省略して＜出力ファイ

ル名＞を指定するときは、＜コマンド＞との間にカンマを2つ(,,)入れます。

コマンドラインの任意の場所に〝;〟を入力すると、それ以降の指定は無視されます。

**例：**　LIB lang -+heap ;

この例では、処理の対象となるライブラリファイルは〝lang.lib〟です。このライブラリファイルの拡張子は〝.LIB〟なので、これを省略して指定しています。

ライブラリに加える編集(コマンド)は、〝-+heap〟で、ライブラリ中のモジュール〝heap〟を新しいモジュールに交換します。このコマンドに従って、ライブラリ中のモジュール〝heap〟をオブジェクトファイル〝heap.obj〟の内容と置き換えます。

コマンドラインの最後にセミコロン(;)がついているので、これよりも後ろのパラメータに関しては、初期設定に従った処理が行われます。このため、リストファイルは生成されず、編集はライブラリファイルに直接反映され、新しいライブラリ(出力ファイル)は作成されません。

**例：**　LIB lang -+heap, lang.1st, lang1.lib

この例は、前例と同様に、ライブラリ〝lang.lib〟のモジュール〝heap〟を置き換えるものですが、リストファイル名と出力ファイル名が指定されています。このため、クロスリファレンスリストファイル〝lang.1st〟を作成します。また、編集結果は、出力ファイル〝lang1.lib〟に収められ、編集の元になったライブラリ〝lang.lib〟は、元の状態のまま残されます。

## ■ LIBのプロンプトでの指定方法

コマンドラインで〝LIB〟だけを入力すると、処理に必要な情報の入力を求めるコマンドプロンプトを表示します。コマンドプロンプトによる入力方法はつぎのとおりです。

1. LIBとタイプし、リターンキーを押すと、つぎのようなプロンプトが表示されます。

   Library name :

2. ここで作成・編集したい、ライブラリファイル名をタイプします。ライブラリファイル名の拡張子が〝.LIB〟のときは、これを省略することができます。ファイル名をタイプしたら、リターンキーを押します。指定したファイルを検索して見つからない場合は、つぎのようなプロンプトを表示します。

   Library file does not exist. Create?

LIB

ここで〝Yes〟を入力すると、新しいライブラリファイルが作成されます。〝No〟を入力すると、MS-DOSに戻ります。

3. ライブラリファイル名の入力が済むと、LIBは、ライブラリに行う編集処理(オペレーション)の指示を求める第2のプロンプトを表示します。

   Operations：

   ライブラリに行う編集操作(コマンド)をタイプして、リターンキーを押します。指示するコマンドが多く、1行で入りきらないときは、その行の最後にアンパサンド記号(&)をタイプして、リターンキーを押します。LIBは、同じコマンドプロンプトを次の行に表示するので、コマンドの続きをタイプします。すべてのコマンドをタイプしたら、リターンキーを押します。ライブラリファイルの整合性(5.2節中の「ライブラリの整合性のチェック」を参照)をチェックするときは、リターンキーだけを押してください。

4. つぎに、LIBは、リストファイル名の入力を求める、第3のコマンドプロンプトを表示します。

   List file：

   クロスリファレンスリストファイルの名前をタイプし、リターンキーを押します。クロスリファレンスリストファイル名は、ユーザーが自由につけることができます。ファイル名の拡張子を自動的につけることはしません。クロスリファレンスリストファイルが不必要なときは、何もタイプせずに、リターンキーを押してください。ライブラリファイルを新規に作成する場合、表示されるコマンドプロンプトはここまでで、LIBはライブラリの作成・編集処理を開始します。

5. すでにあるライブラリファイルを編集する場合は、つづいて出力ファイル名の入力を求める、最後のプロンプトが表示されます。

   Output library：

   ライブラリに編集を行った結果(出力)を収める、新しいライブラリファイル名(出力ファイル名)を入力します。ファイル名拡張子を省略すると、LIBは、自動的に〝.LIB〟という拡張子をつけます。
   出力ファイル名を指定せずにリターンキーだけを押すと、指定したライブラリファイルを編集しますが、この場合は編集の元になったライブラリファイルの拡張子が〝.BAK〟につけかえられます。
   出力ファイル名をタイプしたら、リターンキーを押してください。LIBは、ライブラリの作成・編集処理を開始します。

第 2 のコマンドプロンプト以降の入力で、セミコロン（；）を入力すると、それ以後の処理が、LIB の初期設定に従って行われます。LIB は、セミコロンが入力されると、それ以後のコマンドプロンプトの表示をせずに、ライブラリの編集を開始します。

処理に関係したファイルが、カレントディレクトリ以外にあるときは、ドライブ名、パス名を含めた正しいファイル名を指定してください。

ライブラリファイルのページサイズを指定することができます。この指定は、ライブラリファイル名の指定と共に行います。詳しくは、5.2 節中の「ライブラリページサイズの設定」を参照してください。

**例：**　LIB の表示するコマンドプロンプトに沿った、入力例をつぎに示します。

```
Library File：math
Operations：+sin+cos+atan&
Operations：+exp
List file：
Output library：math1
```

この例では、ライブラリファイル〝MATH.LIB〟が処理の対象になっています。＜コマンド＞では、4 つのオブジェクトモジュール（sin、cos、atan、exp）の追加が指定されています。各モジュールは、SIN.OBJ、COS.OBJ、ATAN.OBJ、EXP.OBJ というファイルに収められています。2 行目のコマンドプロンプトの最後にタイプしてあるアンパサンド記号（&）によって、指定したいコマンドの続きを入力することができます。〝List file：〟プロンプトに対しては何もタイプせずにリターンキーを押しているので、クロスリファレンスリストファイルは作成されません。ライブラリ〝MATH.LIB〟に 4 つのオブジェクトモジュールを追加した結果（新しいライブラリ）は、〝MATH1.LIB〟という名前のファイルに収められます。編集の元になったライブラリ〝MATH.LIB〟は、そのままの内容で保存されます。

## ■ 応答ファイルでの指定方法

ファイル名やコマンドなど、LIB に対する入力だけを収めた〝応答ファイル〞を用意しておき、この応答ファイルの内容に従って処理を進める方法です。このときはつぎのように〝@〞をつけて入力します。

LIB　@<応答ファイル名>

アットマーク(@)は、後ろにつづくファイル名が(ライブラリファイル名ではなく)応答ファイルであることを表します。また、応答ファイルがカレントディレクトリ以外にあるときは、ドライブ名、パス名を含めて指定できます。

応答ファイルの名前は、自由につけることができます。このファイルの内容は、つぎのような形で記述します。

<ライブラリファイル名>[/PAGESIZE：n]
<コマンド>
<クロスリファレンスリストファイル名>
<出力ファイル名>

応答ファイル内の各行は、前項で解説したプロンプトの入力に対応しています。<コマンド>が多く、1 行に収まらないときは、その行の最後にアンパサンド記号(&)をつけて、次の行に継続することができます。パラメータを指定しない場合は、その行を空にしておきます。

応答ファイルを利用する方法で起動すると、コマンドプロンプトとそれに対応する応答ファイルの内容を表示しながら処理を進めます。応答ファイルの内容が不足しているときは、該当するコマンドプロンプトを表示し、ユーザーの入力を待って処理を再開します。

応答ファイル中でも、LIB の初期設定に従った処理を指示する、セミコロン(；)を利用することができます。

**例：**

```
PLIB
+cursor-+heap*stack
cross.lst
```

この例のような内容を持つ応答ファイルでは、処理されるライブラリファイル名は〝PLIB.LIB〞です。コマンドは、モジュール〝cursor〞の追加、〝heap〞の置き換え、〝stack〞のコピーが指示されています。また、クロスリファレンスリストファイル〝cross.lst〞が、作成されます。

## ■ ライブラリの整合性のチェック

ライブラリファイル名だけを指定して、コマンドを指定しないと、LIB は、ライブラリの整合性をチェックします。

LIB は、指定されたライブラリ中のすべてのモジュールがアクセスできるかどうか調べます。調べたライブラリ中になんらかの問題があった場合は、エラーメッセージを表示します。問題がなかったときは、特にメッセージは出力されず、MS-DOS のプロンプトが表示されます。

整合性のチェックは、ライブラリ見出しテーブルと実際の内容に差がないことを確かめるときなどに行います。たとえば、ライブラリファイルを他のディスクからコピーした後など、この整合性のチェックを行って、コピーが正確に行われたか否かを確認するとよいでしょう。

なお、モジュールを追加するときは、事前に整合性のチェックを行うので、モジュールを追加する度に整合性チェックを行う必要はありません。

**例：**　　LIB math；

この例は、ライブラリファイル〝MATH.LIB〟のチェックを行うものです。

## ■ ライブラリページサイズの設定

ユーザーはライブラリのページサイズを設定することができます。ページサイズの設定は、編集・作成するライブラリファイル名の指定と共に行います。指定するときは、/PAGESAIZE スイッチを使って、つぎのように記述します。

/PAGESIZE：<n>

ここで、<n>は希望するページサイズで、16〜32768 の間の 2 のべき乗数を指定することができます。

ページサイズの設定は、ライブラリ中のモジュールの配置（配列）を制御するものです。モジュールは、常に、ページサイズ（バイト単位）の倍数の位置からスタートされるように配置されます。ライブラリを新規に作成するときの LIB の初期設定は、16（バイト）となっています。

ページサイズは、LIB の採用しているインデックス技法と関連しています。この方法では、ページサイズは大きいほど、ライブラリに多くのモジュールを登録することができます。しかし、ライブラリ中の各モジュールは、平均 n/2 バイト（n はページサイズ）の無駄なスペースを含んでいます。したがって、実用上は小さなページサイズの方が有益なことが多く、できるかぎり小さなページサイズを指定した方がよいでしょう。

**例：**　　LIB math/PAGESAIZE：256

この例は、ライブラリ"MATH.LIB"のページサイズを256(バイト)に設定したものです。

## ■ クロスリファレンスリストの生成

クロスリファレンスリストファイル名を指定すると、クロスリファレンスリストを得ることができます。このクロスリファレンスリストには、2種類のリストが収められます。1つはライブラリ中のすべてのパブリックシンボルのリストで、もう1つはライブラリ中のすべてのモジュールのリストです。

パブリックシンボルのリストには、すべてのシンボルがアルファベット順に収められます。各シンボルは、それを参照しているモジュール名の後に表示されます。

**例：**

```
EXIT......error
START.....main
SUM.......add
SUM2......add
```

モジュールのリストには、すべてのモジュールがアルファベット順に収められます。モジュール名は、そのモジュール中で参照されるアルファベット順のパブリックシンボルの後ろに表示されます。

**例：**

```
error    Offset：  00000600H    Code and data size：CH
    EXIT
main     Offset：  00000200H    Code and data size：20H
    START
add      Offset：  00000400H    Code and data size：20H
    SUM                    SUM2
```

## 5.3　LIBのコマンド

<コマンド>の指示に用いる、LIBのコマンドにはつぎのようなものがあります。モジュールの追加・削除・置換はライブラリの編集に用い、コピー・移動は新しいライブラリの作成に関係したコマンドです。

| 機　能 | コマンド |
|---|---|
| モジュールの追加 | ＋ |
| ライブラリの連結 | ＋ |
| モジュールの削除 | − |
| モジュールの置換 | −＋ |
| モジュールのコピー | * |
| モジュールの移動 | −* |

これらのコマンドは、処理の対象となるオブジェクトファイル名・オブジェクトモジュール名の先頭につけて使用します。

LIB

### ■ モジュールの追加

**書　式**

**＋<オブジェクトファイル名>**

**解　説**

追加(＋)コマンドは、オブジェクトモジュールをライブラリに登録するときに使います。<オブジェクトファイル名>は、追加したいモジュールを収めたファイル名です。このファイルの拡張子が".OBJ"のときは、これを省略することができます。ファイルがカレントディレクトリ以外に存在しているときは、ドライブ名やパス名などを含めて指定してください。追加コマンドを示す記号(＋)と、オブジェクトファイル名の間には、スペース(空白)を入れないでください。モジュールは、常にライブラリファイルの後ろへ追加されます。

**サンプル**

```
LIB math +sin.obj
```

この例は、オブジェクトファイル"sin.obj"中のモジュールをライブラリ"math.lib"に追加するものです。

```
LIB ¥lib¥math +cos, list ;
```

この例は、¥libディレクトリのライブラリ"math.lib"へ、オブジェクトファイル"cos.obj"中のモジュールを追加するものです。さらに、クロスリファレンスリストファイル"list"の作成を指定しています。

```
LIB math +A:¥src¥atan;
```

この例は、カレントディレクトリのライブラリ〝math.lib〟へ、ドライブＡ：のサブディレクトリ〝src〟のオブジェクトファイル〝atan.obj〟中のモジュールを追加するものです。

## ■モジュールの削除

**書　式**

-<モジュール名>

**解　説**

削除コマンド(-)は、<モジュール名>で指定されたオブジェクトモジュールを指定されたライブラリファイルから削除します。<モジュール名>は、ライブラリ見出しテーブルに登録されている名前で、正確に指定してください。

なお、LIBはコマンドライン上の順番に関係なく、モジュールの追加に先立って、削除を行います。したがって、モジュールの更新(新バージョンへの変更)は、確実に行われます。

**サンプル**

```
LIB math -sin
```

この例は、ライブラリファイル〝math.lib〟の、モジュール〝sin〟を削除するものです。

```
LIB ¥lib¥math -cos, list;
```

この例は、サブディレクトリ〝lib〟の、ライブラリファイル〝math.lib〟から、モジュール〝cos〟を削除するものです。さらに、編集の終わったライブラリファイルの、クロスリファレンスリストファイル〝list〟を作成します。

```
LIB math +a:¥src¥atan -atan;
```

この例は、ライブラリファイル〝math.lib〟の、モジュール〝atan〟を削除し、ドライブＡ：のディレクトリ〝src〟にあるオブジェクトファイル〝atan〟を追加します。ここでは、同じ名前のモジュールの追加が先に指定されていますが、LIBは、コマンドライン上で指定される順番に関係なく、削除を先に行います。

## ■ モジュールの置換

書　式

-+<モジュール名>

解　説

置換コマンド(-+)は、指定されたモジュールをライブラリファイルから削除するとともに、同じ名前のオブジェクトモジュールファイルをライブラリに追加します。指定する<モジュール名>は、ライブラリ見出しテーブルに登録されている名前で、正確に指定してください。

LIB は、まず指定されたオブジェクトモジュールをライブラリから削除します。次に、カレントディレクトリ中で、削除した<モジュール名>に拡張子〝.OBJ〞のついたファイルを捜し、これを追加します。

サンプル

```
LIB math -+cos ;
```

この例は、ライブラリファイル〝math.lib〞の、モジュール〝cos〞を削除し、オブジェクトモジュールファイル〝cos.obj〞を追加します。

## ■ モジュールのコピー

書　式

*<モジュール名>

解　説

コピーコマンド(*)は、指定された名前のモジュールをコピーし、同じ名前のオブジェクトモジュールファイルを作成します。<モジュール名>は、ライブラリ見出しテーブルに登録されている名前で、正確に指定してください。

コピーしたモジュールが収められるオブジェクトファイル名は、<モジュール名>と同じ名前で、拡張子〝.OBJ〞がつけられます。また、このファイルは、カレントディレクトリに作成されます。

サンプル

```
LIB math *cos ;
```

この例は、ライブラリファイル〝math.lib〞の、モジュール〝cos〞をコピーして、オブジェクトモジュールファイル〝cos.obj〞を作成します。

LIB

## ■ モジュールの移動

**書　式**

-*<モジュール名>

**解　説**

移動コマンド(-*)コマンドは、指定されたモジュールをライブラリファイルから抜き取り、オブジェクトモジュールファイルに収めます。＜モジュール名＞は、ライブラリ見出しテーブルに登録されている名前で、正確に指定してください。

抜き取ったモジュールが収められるオブジェクトファイル名は、＜モジュール名＞と同じ名前で、拡張子〝.OBJ″がつけられます。また、このファイルは、カレントディレクトリに作成されます。また、オブジェクトモジュールファイルに移されたモジュールは、ライブラリから削除されます。

**サンプル**

```
LIB math -* cos
```

この例は、ライブラリファイル〝math.lib″の、モジュール〝cos″を抜き取り、オブジェクトモジュールファイル〝cos.obj″に移します。モジュール〝cos″は、ライブラリから削除されます。

## ■ ライブラリの連結

**書　式**

+<ライブラリファイル名>

**解　説**

追加コマンド(+)を用いると、編集を指定したライブラリへ、指定したライブラリファイル全体を追加することもできます。追加する＜ライブラリファイル名＞の拡張子が〝.LIB″のときは、これを省略することができます。

追加されるライブラリは、編集を指定したライブラリの後ろへ、すでにあるモジュールを変更•削除することなく追加されます。

なお、LIB は、XENIX 方式、インテル方式のライブラリを MS-DOS のライブラリへ追加することも可能です。

**サンプル**

```
LIB math1 +math.lib ;
```

この例は、編集中のライブラリファイル〝math1.lib″へ、ライブラリファイル〝math.lib″を追加するものです。

## 5.4　メッセージ

LIB が表示するメッセージをアルファベット順に解説します。

Cannot access file

ファイルにアクセスできませんでした。

Cannot create extract file : **<ファイル名>**

出力ファイルを作成できませんでした。

Cannot create listing : **<ファイル名>**

リストファイルを作成できませんでした。

Cannot create new library

新しいライブラリを作成できませんでした。

Cannot open response file

応答ファイルをオープンできません。

Cannot open VM.TMP

一時ディスクファイル VM.TMP をオープンできません。

Cannot read from VM

一時ディスクファイルから読み込むことができません。

Cannot rename old library

古いライブラリをリネームできません。

Cannot reopen library

ライブラリを再オープンできません。

Cannot write to VM

一時ディスクファイルに書き込むことができません。

Comma or new line expected

カンマ、または改行記号を入れてください。

Error writing to cross reference file

クロスリファレンスファイルにエラーを記録しました。

Error writing to new library
新しいライブラリにエラーを記録しました。

Incorrect DOS version
DOSのバージョンが違います。動作可能なDOSのバージョンで起動してください。

Insufficient memory
メモリ容量が不足です。

Insufficient stack space
スタック領域が不足です。

Internal failure
内部エラーです。

Invalid library
ライブラリファイルが不正です。

Invalid format
ファイルのフォーマットが不正です。

Invalid library header
ライブラリファイルのヘッダが不正です。

Missing terminator
ターミネータがありません。

Module not in library
モジュールがライブラリ内にありません。

Module redefinition in file ignored
モジュールの再定義は、無視されました。

No more virtual memory
仮想メモリが不足です。

Page size too small
ページサイズが正しくありません。

Page size too small
ページサイズが小さすぎます。

Symbol redefinition in file ignored

シンボルの再定義は、無視されました。

Syntax error

文法エラーです。

Too many symbols

シンボルの数が多過ぎます。

Unexpected EOF on command input

コマンドの入力がEOF(エンドオブファイルマーク)によって終了してしまいました。

Unknown switch

指定したスイッチが不正です。

Write to extract file failed

出力ファイルへの書き込みに失敗しました。

Write to library file failed

ライブラリファイルへの書き込みに失敗しました。

# 第 6 章

# MAKE：プログラムメインテナ

## 6.1 イントロダクション

プログラムメインテナ：MAKEは、マクロアセンブラや高級言語によるプログラムの開発工程の保守・管理を自動的に行うユーティリティです。MAKEは、ソースファイルの更新に伴なって、それに関連するプログラムの更新など必要な処理をすべて自動的に実行します。

MAKEの機能は、いわゆるバッチ処理とは異なり、さらに高度です。メインテナンスの目的とするファイル(オブジェクトファイル、ライブラリファイル、クロスリファレンスリストファイルなど)と、それの元になる関連ファイル(ソースファイル、クロスリファレンスファイル、オブジェクトファイルなど)の作成日付を比較し、目的ファイルの日付が新しい場合にだけ、特定の更新に必要な処理を実行します。さらにMAKEによる処理で生じたファイルの作成日付の食い違い(日付の違い)に関する処理も行います。

MAKEはバッチ処理のように、ファイルが1つでも更新されれば、すべてのファイルのアセンブル、コンパイル、リンクを行うようなことはしません。MAKEを使用することによって、多くのソースファイルを持つプログラムや、完成までに多くのステップを必要とするプログラムの作成で、かなりの手間と時間が節約できます。

MAKE

### ■ メイクファイル

MAKEを使用する場合、実行したい作業とその作業に関連するファイル名を記述した〝メイクファイル〞をあらかじめ作成しておきます。

MAKEはこのメイクファイルを読み込んで、それにしたがってコンパイル、アセンブル、リンクなどの作業を自動的に行います。

メイクファイルは、テキストエディタを用いて作成します。ファイルはつぎのような形式の〝作業記述〞を1つ以上記述したものです。

**<目的ファイル名>：<関連ファイル名 1>　<関連ファイル名 2> …**
**<コマンド>**

<目的ファイル名>

更新を必要とするファイルの名前を指定します。指定できるのは1つのファイルだけで複数指定することはできません。必要であればパス名を含めて指定することもできますが、ドライブ名を指定することはできません。

＜関連ファイル名＞

目的ファイルが関連する(作業に必要となる)ファイルの名前です。必要であればパス名、ドライブ名を含めて指定してください。コマンドは、MS-DOS の外部コマンドです。関連ファイルは複数指定できますが、このときはスペースで区切る必要があります。関連ファイルが 1 行に入りきらない場合、行の終わりに円記号(¥)を入力すれば、次の行に名前を続けることができます。

＜コマンド＞

MS-DOS の外部コマンドで、TYPE や COPY などの内部コマンドは記述できません。コマンドは、複数指定することができますが、各々は必ず新しい行から始め、先頭にタブかスペースを入れる必要があります。コマンドは、目的ファイルが存在しないとき、あるいは目的ファイルの作成日付以後に関連ファイルが修正された場合にだけ実行されます。また、環境変数"PATH"に設定された検索パスによって、コマンドは実行されます。

メイクファイルの中には、このような作業記述をいくつも記述できますが、1 つ記述の最終行と次の記述の間は、1 行以上空けて区切らなくてはなりません。

**注意**　作業記述を書く順番は重要です。MAKE は記述順に指定された処理を行っていくからです。

**例：**

```
startup.obj :   startup.asm
                MASM startup,startup ;

print.obj :     print.asm
                MASM print,print,print,print

print.ref :     print.crf
                CREF print,print

print.exe :     startup.obj print.obj¥lib¥syscal.lib
                LINK startup+print,print,print/map,¥lib¥syscal ;

print.sym :     print.map
                MAPSYM  -l print.map
```

この例では、5 つの目的ファイル作成のための処理が定義されています。各ファイルは、最低 1 つの関連ファイルと 1 つのコマンドを持ちます。作業記述は、目的ファイルの作成順序で入力します。

### ■ MAKEの起動

MAKEはつぎのような書式で起動します。

**MAKE <メイクファイル名>**

＜メイクファイル名＞は、あらかじめ作成しておいたメイクファイルの名前です。メイクファイル名は自由につけることができますが、通常、開発するプログラムと同じ名前を付けて、拡張子は付けません。

MAKEは起動されると、作業記述を順に調べます。指定された目的ファイルの日付がその関連ファイルの日付より新しかったり、該当する目的ファイルがなかったすると、MAKEは指定されたコマンドを表示して実行します。そうでない場合は、次の作業記述に移ります。

指定したメイクファイル、または作業記述中の関連ファイルが見つからなかったときは、MAKEはつぎのようなメッセージを表示します。

```
make：<メイクファイル名>-file not found
```

**例：** make test

## 6.2 MAKEの動作例



つぎのようなメイクファイルを例にとって、MAKEの動作を解説します。

```
test.obj：    test.asm
        MASM test,test,test,test

test.ref：    test.crf
        CREF test,test

test.exe：    test.obj¥lib¥math.lib
        LINK test,test,test/map,¥lib¥math

test.sym：    test.map
        MAPSYM  -l test.map
```

このメイクファイルのファイル名を〝TEST〞とすると、実行するときはつぎのように入力します。

```
MAKE test
```

MAKEが行う各ステップの処理をつぎに示します。

1. 〝test.asm〞の修正日付と〝test.exe〞の日付を比較します。〝test.exe〞の日付の方が新しいか、または存在しない場合、MAKEは、つぎのコマンドを実行します。

   ```
   MASM test,test,test,test
   ```

   それ以外の場合は、MAKEは、次の作業記述へ進みます。

2. 〝test.ref〞と〝test.crf〞の日付を比較します。〝test.ref〞の日付の方が新しい場合、MAKEはつぎのコマンドを実行します。

   ```
   CREF test,test
   ```

3. 〝test.exe〞と〝test.obj〞、および〝test.exe〞と〝math.lib〞との日付を比較します。〝test.exe〞の日付がどちらかのファイルより新しい場合、MAKEはつぎのコマンドを実行します。

   ```
   LINK test,test,test/map,¥lib¥math.lib
   ```

4. 〝test.sym〞と〝test.map〞の日付を比較します。日付が新しい場合は、つぎに示すコマンドを実行します。

   ```
   MAPSYM -l test.map
   ```

〝test.asm〞が最初に作成されたときは、MAKEはすべてのコマンドを実行します。これは、メイクファイル中のすべての目的ファイルがないためです。どの関連ファイルも変更しないでMAKEを再度起動させると、MAKEはどのコマンドも実行しません。
ライブラリファイル〝math.lib〞を変更してその他の変更を行わなかった場合、LINKコマンドを実行します。これは、〝test.exe〞の日付が〝math.lib〞より新しいためです。さらにMAPSYMコマンドも実行しますが、これは〝test.map〞がLINKによって作成されるためです。

MAKEは、このようにファイルの作成日付を判断しながら処理を行いますので、修正されたファイルに関する処理のみを行うことができ、処理時間の短縮を図ることが可能となります。

## 6.3 メッセージ

MAKEが表示するメッセージをアルファベット順に解説します。

Access denied

リードオンリーファイルに書き込むことはできません。

Bad format

MAKEファイルのフォーマットが不正です。

File already exists

作成すべきファイルと同名のファイルが既に存在しています。

File not found

処理の対象となるファイルが見つかりません。

Invalid data

MAKEファイル内のデータが不正です。

Invalid drive

ドライブの指定が不正です。

Make：unexpected end of file

MAKEファイルが完全ではありません。

Make: out of memory

メモリ不足です。

Too many open files

オープンするファイルの数が多過ぎます。

# 第7章

# EXE2BIN：バイナリファイルコンバータ

## 7.1 イントロダクション

バイナリファイルコンバータ：EXE2BIN は、EXE 形式(拡張子が".EXE")のプログラムファイルに加工を施して、バイナリファイルを作成するためのユーティリティです。EXE2BIN は、一般に EXE 形式のファイルからヘッダを取り除いて COM 形式のファイルに変換するのに用いますが、その他にも、ROM プログラムなどのような絶対アドレス上に置かれるプログラムを生成するために使用することもできます。

## 7.2 EXE2BIN の起動と使い方

EXE2BIN は、つぎのような書式で起動します。

EXE2BIN <ファイル名 1> [<ファイル名 2>]

<ファイル名 1>

リンカ(MS-LINK)で作成されたエラーのない EXE 形式ファイルです。ファイルの常駐部あるいはファイルのコードとデータ部分が 64K バイト以下でなければなりません。また、スタックセグメントが存在してはなりません。<ファイル名 1>の拡張子を省略した場合は、EXE 形式ファイルと解釈します。

<ファイル名 2>

BIN(バイナリ)形式に変換された結果を格納するファイル名です。<ファイル名 2>全体を省略すると、<ファイル名 1>とおなじファイル名に拡張子".BIN"をつけたものとなります。<ファイル名 2>の拡張子のみを省略したときは、指定したそのファイル名に拡張子".BIN"をつけたものとなります。

EXE2BIN は、処理の対象となる EXE 形式のファイルが持っている条件(プログラムの開始アドレスやヘッダの内容など)により、つぎに述べるような 2 種類の変換を行います。

### 1．COM 形式ファイルの作成

EXE 形式のプログラムファイルを COM 形式ファイルに変換する場合、該当するプログラムはつぎのような条件を満たしている必要があります。

- EXE 形式ファイルのヘッダ部分に、セグメントの設定を要求するリロケート情報がない
- プログラムの開始アドレスが、END 疑似命令によって先頭からオフセット 100H 番地に定義されている
- セグメントアドレスを参照する FAR コール／ジャンプがない
- スタックセグメントがない
- プログラム全体(常駐部、またはコードとデータ部分)が 64K バイト以下である

これらの条件を満たしていれば、そのプログラムはパラグラフ単位でリロケータブルですから、セグメントアドレスを参照している部分をリロケートする必要はなく、したがってヘッダを取り除いても実行には影響を与えません。

EXE2BIN でこのようなプログラムを変換すると、EXE 形式ファイルのヘッダ部分だけでなく、プログラム本体部分の最初の 256(100H)バイトも削除します。

こうして作成されたファイルは、その拡張子〝.BIN″を〝.COM″にリネームするだけで、COM 形式のコマンドファイルとしてそのまま MS-DOS で実行できるようになります。サイズが縮小した分だけディスクからのロードが速くなり、リロケートが不要な分だけ実行開始までの時間を短縮することができます。

### 2．BIN 形式ファイルの作成

COM 形式に変換するための条件を備えていないプログラムの場合、EXE2BIN は単純なバイナリ変換を行います。この場合でも、該当する EXE 形式のプログラムファイルはつぎのような条件を満たしている必要があります。

- プログラムの開始アドレスが、ないか、または先頭番地に定義されている
- スタックセグメントがない
- プログラム全体(常駐部、またはコードとデータ部分)が 64K バイト以下である

EXE 形式のファイルのヘッダ部分に、セグメントの設定を要求するリロケート情報がなければ、EXE2BIN はそのファイルをそのままバイナリ変換します。

ただし、リロケート情報がある場合、EXE2BIN は、プログラムをロードすべきセグメントアドレスを入力するよう要求してきます。

```
Fix-ups needed - base segment(hex) :
```

この結果生成されたプログラムは、絶対アドレスに配置されたメモリイメージの形態となります。したがって、プログラムは、ユーザーのアプリケーションプログラムでロードするような場合のみ実行することができます。コマンドプロセッサ(COMMAND.COM)には、このようなプログラムを正しくロードする機能はありませんので注意してください。

以上2つの条件のどちらにもあてはまらないか、COM 形式ファイルの条件を満たしてはいるがセグメントの設定がある場合などは、変換できません。この場合、EXE2BIN はエラーメッセージを表示して実行を終了します。

## 7.3 メッセージ

EXE2BIN が表示するメッセージをアルファベット順に解説します。

Access denied

リードオンリーファイルに書き込むことはできません。

Amount read less than size in header.

読み込んだファイルのサイズが、ヘッダ中に記録されているサイズより小さいです。何らかの原因でファイルが不正となっています。

File creation error

システムファイル、または隠しファイルに書き込みをしようとしました。ファイルの属性を確認してください。

File cannot be converted

ファイルをバイナリ変換することはできませんでした。

File not found

指定されたファイルが見つかりません。

Incorrect DOS version

DOS のバージョンが違います。動作可能な DOS のバージョンで起動してください。

Insufficient disk space

ディスク容量に十分な空きがないため、書き込むことができません。

Insufficient memory

メモリ容量が足らないため、プログラムを正常に実行できません。

WARNING - Read error on EXE file.

EXE 形式のファイルの読み込み中に、エラーが発生しました。

# 第8章

# DEBUG：デバッガ

## 8.1 イントロダクション

デバッガ：DEBUGは、メモリやレジスタの操作、プログラムのトレースや逆アセンブルを行うだけでなく、ディスク内のデータをレコード単位で読み書きしたり、EMSメモリ(拡張メモリ)のマッピングを行ったりすることが可能な、強力なデバッガです。

ただし、SYMDEB(シンボリックデバッガ)とは違い、シンボルファイルを用いたデバッグはできませんので注意してください。

## 8.2 DEBUGの起動

DEBUGを起動すると、DEBUGのプロンプト(-)が表示され、DEBUGのコマンドを入力できる状態になります。DEBUGは、つぎのような書式で起動します。

DEBUG <実行ファイル名> <引数リスト>

<実行ファイル名>

デバッグする実行可能ファイルのファイル名を指定します。

<引数リスト>

<実行ファイル>に与える引数を記述します。

**例：** DEBUG file.exe

この例のように実行ファイルのみ指定した場合、DEBUGは、使用可能なメモリの最下位セグメントの先頭から100H(256)バイトのプログラム用のヘッダを作成し、100H番地以降にファイルをロードします。ただし、EXE形式のファイルの場合は、ファイルのヘッダ部分で定義されたアドレスにロードします。

さらに、ファイルの大きさ(バイト単位)をBX：CXレジスタにセットし、セグメントと他のレジスタをファイルのヘッダ部分で定義された個々の値にセットします。

**例：**　DEBUG file.exe test.dat/m/b

この例では、"test.dat/m/b"をプログラム用のヘッダにコピーし、それから"file.exe"をロードします。"test.dat/m/b"は"file.exe"の引数になります。

すべてのパラメータを省略して、単に"DEBUG"と入力して起動することもできます。起動後、DEBUGのプロンプトが表示され、DEBUGのコマンドが入力できます。

**例：**
```
A>debug
-
```

## 8.3　コントロールキーの使用

DEBUGを実行中、誤りの修正やコマンドの停止には、MS-DOSユーザーズリファレンスマニュアルで述べられているコントロールキャラクタやテンプレート機能を用いることができます。

DEBUGは[CTRL]+[C]キーを押すと、実行中のコマンドを停止してプロンプトを表示します。[CTRL]+[S]キーを押すと、DEBUGはコマンド出力を一時的に中断します。

また、Goコマンドでプログラムを実行中、[CTRL]+[C]キーで停止することもできますが、これはプログラムが入出力の処理をしているときに限られます。

## 8.4　コマンドとパラメータの表記

コマンドの後には、パラメータを必要とするものもあります。複数のパラメータを指定する場合は、カンマ(,)またはスペースでパラメータを区切ります。コマンド名とパラメータについては、大文字と小文字の区別は行われません。

以後の項では、コマンドのパラメータについて詳しく説明します。

### ■数　値

ポート番号、ドライブ番号、レコード番号、コマンドの繰り返し回数などを指定するための4桁までの16進数です。

**例：**
```
1010
  20
```

## ■ アドレス

アドレスは、"＜セグメント＞：＜オフセット＞"という形式で指定します。セグメントとオフセットは、数またはレジスタ名で指定します。セグメントを省略すると、デフォルトのセグメントアドレスが与えられます。デフォルトのセグメントは、Assemble コマンド、Go コマンド、Load コマンド、Unassemble コマンド、Write コマンドの場合は CS レジスタ、それ以外のコマンドの場合は DS レジスタになります。

例：　CS：0100
　　　04BA：IP

## ■ アドレスのレンジ指定

アドレスのレンジ(範囲)を指定するには、つぎのような 2 種類の方法があります。

書式 1：　開始アドレスと終了アドレスを指定する方法

**＜開始アドレス＞　＜終了アドレス＞**

この書式では、＜開始アドレス＞と＜終了アドレス＞をスペースで区切って指定します。＜終了アドレス＞を省略した場合、128 バイトの範囲になります。＜終了アドレス＞にはセグメントの指定はできず、この場合そのセグメントは＜開始アドレス＞で指定したものとおなじになります。

例：　CS：100 110

書式 2：　開始アドレスと長さを指定する方法

**＜開始アドレス＞ L＜値＞**

この書式では、コマンドで処理する範囲は、＜開始アドレス＞から＜値＞で示される長さ(バイト数)の範囲までとなります。

例：　CS：100 L100

## ■ ストリング

ストリングは、クォーテーション(”または’)で囲まれた任意の長さの文字列です。ストリングには、シングルクォーテーション(’)かダブルクォーテーション(”)を含めることもできます。ただし、始まりのクォーテーションと終わりのクォーテーションは同じものでなくてはなりません。ストリング中にさらにクォーテーションを入れるときは、そのクォーテーションを 2 個つづけて入力するか、別の種類のクォーテーションを用います。

**例：**　'This is a string.'
''This is a string.''
'This ''string'' is okay.'
''This'' ''string'' ''is okay.''
'This ''string'' is okay.'
''This 'string' is okay.''

## 8.5　DEBUGのコマンド

DEBUGのコマンドは、EMS関連のコマンドを除き、基本的にはSYMDEBの同名コマンドと同等の機能を持っています。これらのコマンドは、つぎの表で〝同等なSYMDEBのコマンド〟の欄に示してありますので、第4章「SYMDEB：シンボリックデバッガ」を参照してください。

| コマンド名 | 書式 | 同等なSYMDEBのコマンド | 機　能 |
|---|---|---|---|
| Assemble | A | A | アセンブル |
| Compare | C | C | 比較 |
| Dump | D | DB | ダンプ |
| Enter | E | EB | データの入力 |
| Fill | F | F | フィル |
| Go | G | G | 実行 |
| Hex | H | H | 16進計算 |
| Input | I | I | ポート入力 |
| Load | L | L | ロード |
| Move | M | M | 移動 |
| Name | N | N | 名前のセット |
| Output | O | O | ポート出力 |
| Quit | Q | Q | DEBUGの終了 |
| Register | R | R | レジスタの表示と設定 |
| Search | S | S | サーチ |
| Trace | T | T | トレース |
| Unassemble | U | U | 逆アセンブル |
| Write | W | W | 書き込み |
| Allocate Expanded Memory | XA | | EMSメモリのアロケート |
| Deallocate Expanded Memory | XD | | EMSメモリのデアロケート |
| Map Expanded Memory Pages | XM | | EMSメモリのマッピング |
| Display Expanded Memory Status | XS | | EMSメモリのステータス表示 |

なお、SYMDEBとDEBUGの機能上の違いにより、DEBUGではシンボルファイルを用いたデバッグを行うことができないため、コマンド中のパラメータとして、つぎに示すものを用いることはできません。

・レジスタ名以外のシンボル
・16進数以外の数値
・行番号
・式

したがって、参照したSYMDEBのコマンドの解説中、またはサンプル中で示されているパラメータを、そのまま使用することのできない場合がありますので注意してください。

コマンド中で、パラメータを省略したときは、デフォルトや直前に行ったコマンドの結果のレジスタ値が用いられます。

この節では、DEBUGでサポートされているEMSメモリ関連のコマンド(XA、XD、XM、XS)についてのみ解説します。

# XA

## Allocate Expanded Memory

## EMSメモリの論理ページをEMSハンドルにアロケートする

**書　式**

XA <論理ページ数>

### 解　説

EMSメモリを＜論理ページ数＞分だけ確保し、それらのページにEMSハンドルをアロケート（割り付け）します。EMSハンドルは16進数の数値で返されます。＜論理ページ数＞は1ページ当たり16Kバイトとして指定します。

**サンプル**

```
-XA 2
ハンドル 000E が作成されました.
```

この例では、2ページ分のEMSメモリのアロケートを要求した結果、それが利用可能であったために、EMSハンドルとして〝000E〞が返されています。

# XD Deallocate Expanded Memory

## EMS ハンドルにアロケートされた EMS メモリの論理ページを開放する

**書式** XD <EMS ハンドル>

### 解説

指定した<EMS ハンドル>にアロケート(割り付け)されている EMS メモリの論理ページをデアロケート(開放)します。

**サンプル**

```
-XD 000E
ハンドル 000E を解放しました.
```

この例では、ハンドル 000E にアロケートされた EMS メモリの論理ページの開放を要求し、それが成功したことを表しています。

DEBUG

# XM　Map Expanded Memory Pages

## EMS ハンドルにアロケートされている論理ページを物理ページにマッピングする

**書　式**　XM <論理ページ番号> <物理ページ番号> <EMS ハンドル>

### 解　説

指定した＜EMS ハンドル＞にアロケート(割り付け)されている EMS メモリの論理ページを、システムメモリ内にある物理ページにマッピングします。＜論理ページ番号＞には、＜EMS ハンドル＞にアロケートされている全論理ページ中の番号を指定します。＜物理ページ番号＞には、全物理ページ中の番号を指定します。

なお、物理ページがシステムメモリのどのセグメントに対応しているかを知るには、XS(Display Expanded Memory Status)コマンドを使用してください。

**サンプル**

```
-XM 5 2 3
論理ページ 05 が物理ページ 02 にマップされました.
```

この例では、ハンドル 0003 にアロケートされた EMS メモリの論理ページ 05 を、物理ページ 02 にマッピングしています。マッピングが成功したため、DEBUG はその旨をメッセージで返しています。

# XS

## Display Expanded Memory Status

### EMS メモリのステータスを表示する

**書　式**　XS

## 解　説

EMS メモリの各種状況を表示します。表示項目はつぎのとおりです。

- 現在使用されているハンドルとそれにアロケートされた EMS メモリの論理ページ数
- 物理ページとそれに対応するフレームセグメントのアドレス
- 現在アロケートされている EMS メモリの論理ページ数と、使用可能な論理ページの総数
- 現在アロケートされている EMS ハンドル数と、使用可能な EMS ハンドルの総数

**サンプル**

```
-XS
ハンドル 0001 には 0002 ページが割り当てられています.
物理ページ 00 = フレームセグメント C000
物理ページ 01 = フレームセグメント C400
物理ページ 02 = フレームセグメント C800
物理ページ 03 = フレームセグメント CC00
    80 個の EMS ページから 02 ページが割り当てられています.
    FF 個の EMS ハンドルから 02 個のハンドルが割り当てられています.
```

この例は、つぎのような状況を表しています。

EMS ハンドル 01 に EMS メモリの論理ページ 2 ページ分がアロケートされている
物理ページ 00 はフレームセグメント C000 に対応
物理ページ 01 はフレームセグメント C400 に対応
物理ページ 02 はフレームセグメント C800 に対応
物理ページ 03 はフレームセグメント CC00 に対応
現在アロケートされている論理ページ数は 2、使用可能な論理ページの総数は 80
現在アロケートされている EMS ハンドル数は 2、使用可能な EMS ハンドルの総数は FF

DEBUG

## 8.6　メッセージ

DEBUGは、実行できないコマンドや不正なパラメータを見つけると、エラーメッセージを表示します。

通常は、発生したエラーに対して、記号(^)と"エラー"というメッセージを表示します。記号(^)は、コマンドライン中のエラーの位置を示すものです。つぎの例は、Dumpコマンドの値の範囲が適切でないことを示しています。

**例：**
```
d cs：100 0
          ^エラー
```

またDEBUGは、エラーメッセージ以外にも、コマンドの実行状況をくわしく伝えるメッセージを表示します。つぎに、それらのメッセージとその意味を示します。

**DOS のバージョンが違います.**

動作可能なDOSのバージョンで起動してください。

**EMS エラーです.**

EMSに関するコマンドを実行した際、何らかのエラーが発生しました。

**EMS が組み込まれていません.**

EMSドライバが組み込まれていません。

**EMS ハードウェア／ソフトウェアのエラーです.**

EMSメモリボード、またはEMSドライバが異常です。

**EMS パラメータが無効です.**

EMSに関するコマンドのパラメータに誤りがあります。

**EXE と HEX ファイルには書込みできません.**

EXE形式とHEX形式のファイルを、ディスクに書き込むことはできません。

**<W>書き込みコマンドエラー.**

ファイルの書き込み中にエラーが発生しました。

**xx バイト書込み中.**

xxバイト分、ディスクに書き込みました。

**空きページ数を越えています.**

指定したページ数が残りのページ数を超えています。

**アクセスは拒否されました.**

リードオンリーファイルに書き込むことはできません。

**エラー, EXE または HEX ファイルにエラーがあります.**

ロードした EXE 形式または HEX 形式のファイルが、正しくありません。

**実行できませんでした.**

実行したプログラムは、異常終了しました。

**出力装置の PRN をオープンできません.**

プリンタが接続されていないか、またはプリンタの電源が入っていません。

**全ページ数を越えています.**

アロケートしたページ数を超えて指定しました。

**装置名が正しくありません.**

出力装置名が正しくありません。

**ディスクがいっぱいです.**

ディスク容量に十分な空きがないため、書き込むことができません。

**転送先が定義されていません.**

転送先のアドレスが指定されていません。

**ドライブの指定が違います.**

指定されたドライブ番号が正しくありません。ドライブ番号は、ドライブ A：なら 0、ドライブ B：なら 1、ドライブ C：なら 2…というように指定します。

**パラメータエラーです.**

指定されたパラメータの数が間違っているか、入力された値が指定可能な範囲内にありません。

**ハンドル<XX>には<XX>ページが割り当てられています.**

同一の EMS ハンドルに異なるページを割り当てることはできません。

**ハンドルがいっぱいです.**

使用可能な EMS ハンドルが残っていません。

**ハンドルが見つかりません.**

指定された EMS ハンドルが見つかりませんでした。

**ファイルが作れません.**

システムファイル、または隠しファイルに書き込みをしようとしました。ファイルの属性を確認してください。

**物理ページの範囲を超えました.**

指定した物理ページの番号が、指定可能な範囲外です。

**プログラムは正常に終わりました.**

実行したプログラムは、正常終了しました。

**メモリエラーが発生しました.**

パリティエラーか、または非実装アドレス領域へのアクセスによりメモリエラーが発生しました。

**論理ページの範囲を超えました.**

指定した論理ページの番号が、指定可能な範囲外です。

# 索引

## 記号

## 数字



## A



## B



## C



## D



## E

## サ

## タ



## ナ



## ハ

## マ



## ヤ



## ラ

## ワ


NEC